

مهندس، أستاذة الحسني
مدير نظم المعلومات بمركز أبحاث الأذن
جامعة كوزيايا الرلايات المتحدة الأمريكية

اطلب مع الكتاب:
القرص المدمج على الأشرطة وصول المميزات

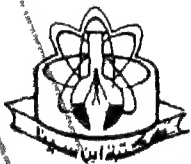
برمجة الرسم بلغة سي++

C++ Graphics

باستخدام المترجم بورلاند سي++ أوتيربوس سي++
على الكمبيوتر الشخصي IBM والكمبيوترات المتوافقة معه

يضمن الكتاب

- الدوال الأساسية للرسم بلغة سي للوصلة البيئة BGI
- بناء الشرائح في البرامج التطبيقية.
- الرسومات الهندسية والأشكال الهندسية:
- الرسم بالفركتالات (Fractals).
- الكتابة بالجمع الكبير بالبرنظات المختلفة.
- الأشكال على الشاشة
- في موضوعات أخرى.



Bibliotheca Alexandrina

مهندس رسالة الحسيني
مدير نظم المعلومات بمركز أبحاث الأردن
جامعة كونيغزيا الولايات المتحدة الأمريكية

برمجة الرسم بلغة سي++

C++ Graphics

باستخدام المترجم بورلاند سي++ أوتيربوس سي++

على الكمبيوتر الشخصي IBM والكمبيوترات المتوافقة معه

يُضمن الكتاب

- الدوال الأساسية للرسم بلغة سي للرسم البياني BGI
- الرسم بالفراكتالات (Fractals).
- بناء الشرائط في البرامج التطبيقية.
- الكتابة بالحجم الكبير بالبرنطيات المختلفة.
- الرسومات الهندسية والأشكال المجسمة: مبادئ وتحريك الأشكال على الشاشة.
- برمجة الفأر الإلكتروني وموضوعات أخرى.

اطلب مع الكتاب: القرص الممّوى على الأمثلة وحلول التمرينات

مكتبة ابن سينا

للنشر والتوزيع والتصدير

٧ شارع محمد قنيد - جامع النج - النزهة

مشرقة - الناعم - ٢٤٧٨٢٢ فاكس ٤٤٨-٤٨٢

وكلاء النوزج

السعودية

مكتبة التامى

الرياض ات ٤٣٥٢٧٦٨ فاكس ٤٣٥٥٩٤٥ طابا جدة ت ٦٥٢٢٠٨٩
القصيم - بريدة ات ٣٢٣١٤٣٤ - المدينة المنورة - ت ٨٢٤٢٧٧٥
ص.ب. ٥٠٦٤٩ - ١١٥٣٣ الرياض

المغرب

دار المعرفة

40 شارع فيكتور ميكلو - الدار البيضاء
ص.ب. 4150 ☎ 300562 - 309520

المكتبة السلفية

12 حي الدخيلة - زقاق الإمام الشافعي - الدار البيضاء
☎ 307643

الإمارات

دار الفضيلة

دبي - ديرة - ص.ب. ١٥٧٦٥ ات ٦٩٤٩٦٨ فاكس ٦٩٤٩٧٦

البحرين

دار الحكمة

ص.ب. ٢٣٨٧٥ هاتف ٢٣٦٠٣٢

جميع الحقوق محفوظة للناشر



كلمة الناشر



تتشرف دار ابن سينا بتقديم هذا الكتاب عن برمجة الرسم بلغة سي++ (باستخدام المترجم بورلاند سي++ أو تيربو سي++) . وبهذا الكتاب فلننا نستكمل ركناً أساسياً في مكتبة البرمجة ولا سيما بلغة سي++ وهي اللغة السائدة في التسعينيات .

ويتناول هذا الكتاب دوال الرسم التي تضمنتها لغة سي++ من خلال أمثلة شائعة وتطبيقات مختلفة ، كما يتناول برمجة الفأر الإليكترونى (mouse) فى بيئة الرسم .

وبذلك فإن الكتاب يقدم للمبرمجين العرب حزمة من الأدوات التى تساعدهم فى بناء برامجهم على الوجه الأكمل .

نأمل أن يحقق الكتاب ما نرجوه من فائدة ..

والله ولى التوفيق ..،

مهندس/مصطفى عاشور

كلمة المؤلف



رُب صورة خير من مقال !

لا يستغنى مبرمج هذه الأيام
عن تدعيم برنامجهِ التطبيقى
بالرسم والألوان والخطوط ذات
الأشكال والأحجام المتنوعة . وقد
تكون الصورة فى أحيان كثيرة
أكثر بلاغة من الشرح الكلامى
المطوّل .

ولا عجب أن يُقبل مستخدموا الكمبيوتر على :

”نوافذ ميكروسوفت“ (Microsoft Windows) ، ويهاجرون إليها
كبديل لنظام التشغيل ”دوس“ (DOS) ؛ فهى بيئة الرسم والألوان
والأشكال الحية .

أما بالنسبة للمبرمج فإن بيئة النوافذ ومثيلاتها من البرامج النابضة
 بالحياة ، ليست إلا برنامجاً مكتوباً بلغة سى++ . فلغة سى++ عامرة
 بإمكانات الرسم التى تبدأ من الأشكال الهندسية البسيطة ، إلى بناء
 النوافذ والقوائم التى نتعامل معها باستخدام الفأر كما الأزرار ، ثم تصل
 فى النهاية إلى مكتبة من الفصائل (classes) يستخدمها المبرمجون فى
 بناء برامج مشابهة للنوافذ أو لبرنامج المترجم بورلاند سى++ أو
 غيرها من البرامج النابضة بالحياة .

وهذا الكتاب يساعدك أن تضع قدمك على الطريق في عالم برمجة
الرسم ، فهو يبدأ معك من البدايات الأولى ويقدم لك الكثير من الأدوات
التي تدعم بها برامجك فتمنحها قوة التأثير وبلاغة التعبير .
والله ولي التوفيق ...،

مهندس/اسامة الحسيني
الولايات المتحدة في يوليو ١٩٩٣

تنويه

© إن البرامج الواردة بهذا الكتاب تستخدم مكتبة الرسم الخاصة بلغة سي وهي لا تختلف عن مكتبة الرسم التي جاءت مع المترجم بورلاند سي++ أو تيربو سي++ .

ومع ذلك فإن البرامج مكتوبة بلغة سي++ وتستخدم الرخص والوسائل الجديدة التي قمتها اللغة . فإذا أردت العودة إلى لغة سي (وهذا مستبعد) فعليك بإجراء بعض التعديلات على البرامج حتى يقبلها مترجم لغة سي (انظر الملحق ج) .

© يحتوى الباب الثامن على برامج مكتوبة باستخدام الفصائل (classes) وهي القضية الأساسية للغة سي++ ولا يجوز استخدامها مع لغة سي .

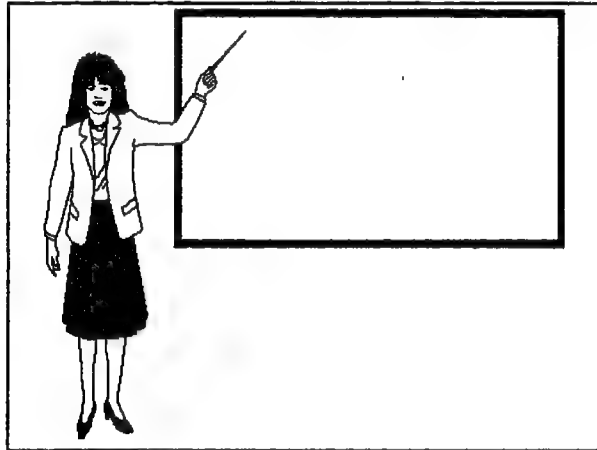
© للاطلاع على مبادئ لغة سي++ يُرجع للجزء الأول من كتابنا :

سي++/سي++ للنوافذ/البرمجة الموجهة نحو الأهداف

© على المبرمجين الذين لم يسبق لهم القراءة باللغة العربية في مجال الكمبيوتر أو لغة سي++ الاستعانة بالملحق (أ) الذى يحتوى على المصطلحات المعربة في هذا المجال .

الباب الأول

جولة التعارف



مفتتح

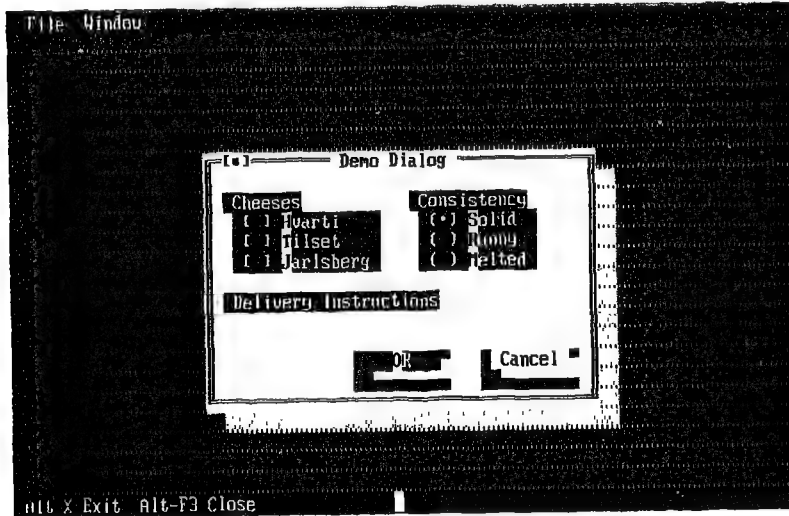
يفضل كثير من المبرمجين أن يكون التعارف بينهم وبين أى موضوع جديد تعارفاً ساخناً ، فيدخلون مباشرة إلى قلب الموضوع ويجربون الدوال المختلفة والبرامج ؛ وذلك بدلاً من المقدمات الطويلة .

وفى هذا الباب سوف يكون هذا هو أسلوبنا حيث نستكشف بيئة الرسم استكشافاً سريعاً من خلال برامج قصيرة ، ثم تأتى التفصيلات تلقائياً وفى موعدها المناسب .

(١ - ١) الرسم فى لغة سى و سى++

لم يتضمن القياس "ANSI" للغة سى عبارات الرسم ، والسبب فى ذلك أن الرسم يعتمد على المعدات المستخدمة اعتماداً كبيراً ؛ والمفترض فى لغة سى القياسية أنها يمكن تنفيذ عباراتها على أى كومبيوتر بحيث تكون لغة منقولة (portable) . ومع ذلك فلا غنى للمبرمج — مع أى لغة من اللغات — عن استخدام الرسم . فالرسم لا يشمل فقط الدوائر والخطوط والأشكال المضلعة ولكنه وسيلة فعالة فى إنشاء القوائم (menus) والنوافذ (windows) التى تجعل البرنامج أليفاً للمستخدم سهل التنفيذ .

وناهيك عن بيئة نوافذ ميكروسوفت (Microsoft Windows) التى تعتمد اعتماداً كلياً على الرسم ، ففى بيئة نظام التشغيل "دوس" يمكنك أيضاً رسم القوائم والنوافذ كما فى الشكل التالى :



شكل (١)

وتحتوى الشاشة فى الشكل التالى على سطر للقائمة نرى فيه اختيارين :
Window, File . ويؤدى أحد الاختيارات إلى فتح قائمة مدلاة
(pull-down menu) ، كما يؤدى أحد الاختيارات بالقائمة المدلاة إلى نافذة
حوار كالموضحة بالشكل .

وقد كان المتبع فى بداية استخدام لغة سي مع الكمبيوتر الشخصى أن يقوم
المبرمج ببناء مكتبته الخاصة من دوال الرسم بادئاً من روتينات الخدمة الخاصة
بالفيديو (ROM BIOS VIDEO SERVICES) . ولكن الطرازات الحديثة
من تيربو سي ، وميكروسوفت سي قد أصبحت تحتوى على مكتبات هائلة
من دوال الرسم يمكن للمبرمج أن يعتمد عليها اعتماداً كلياً .

ويطلق على مجموعة دوال الرسم المتضمنة مع المترجم تيربو سي أو بورلاند
سي الوصلة البيئية BGI اختصاراً للعبارة :

Borland Graphics Interface

ومع تقديم لغة سي++ فقد تضمنت حزمة المترجم بورلاند سي++ مكتبة
جديدة من الفصائل يطلق عليها إجمالاً الاسم : Turbo Vision وهى عبارة
عن الفصائل التى أنشأها مبرمجو شركة بورلاند أثناء إعداد البيئة المجهزة
للمترجم (IDE) بنوافذها وقوائمها ، وبهذا فقد أصبح فى إمكان المبرمج أن
ينشئ لبرامجه بيئة مشابهة ، فكل ما عليه أن يرث فصائل شركة بورلاند ويخلق
منها الأهداف المناسبة . وسوف ينصب الاهتمام فى هذا الكتاب على الوصلة
البيئية للرسم (BGI) وما تقدمه للمبرمجين من إمكانيات .

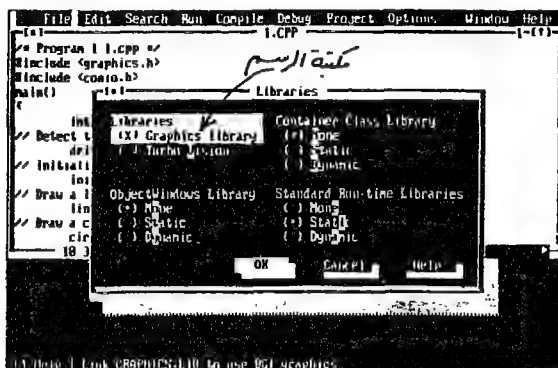
(١ - ٢) إعداد المترجم لبرامج الرسم

قبل تنفيذ أى برنامج من برامج الرسم لابد من إعداد المترجم لربط البرامج
بمكتبة الرسم ، وفقاً للخطوات الآتية :

① المترجم بورلاند سي++ :

— اختر من القائمة العلوية كلمة Options .

- من القائمة المُدلّاة التي تظهر ، اختر كلمة **Linker**
- من النافذة التي تظهر ، اختر كلمة **Libraries**
- ضع علامة X أمام اسم مكتبة الرسم (**Graphic Library**) وذلك بالضغط على زر المسافة الخالية (**space bar**) . لاحظ أن تكرار الضغط على زر المسافة يؤدي إلى إظهار أو إخفاء العلامة X . انظر الشكل التالي :



شكل (٢)

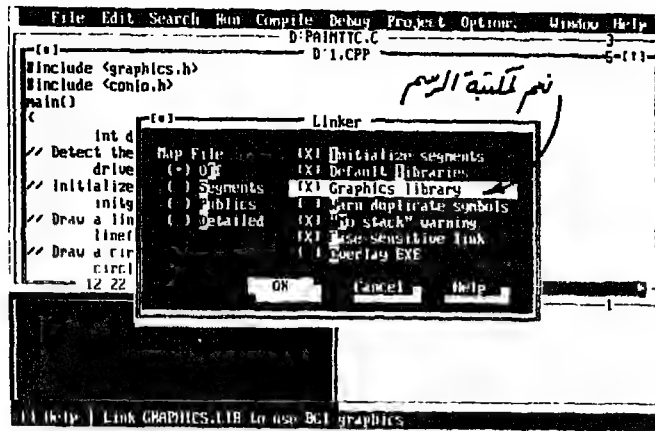
إعداد المترجم بورلاند سي++ لبرامج الرسم

© المترجم تيربو سي++ :

أما مع المترجم تيربو سي++ فإن الخطوات المطلوبة تختصر إلى الآتي :

- اختر من القائمة العلوية كلمة **Options**
- اختر من القائمة المُدلّاة كلمة **Linker** فتظهر النافذة الموضحة بالشكل التالي .

— حرك المؤشر باستخدام الزر **TAB** حتى تصل إلى مكتبة الرسم (**graphic library**) ثم ضع العلامة X أمامها بالضغط على زر المسافة الخالية . لاحظ أن تكرار الضغط على زر المسافة يظهر العلامة X ويخفيها . انظر الشكل التالي :



شكل (٣)

إعداد المترجم تيريو سي++ لتطبيقات الرسم

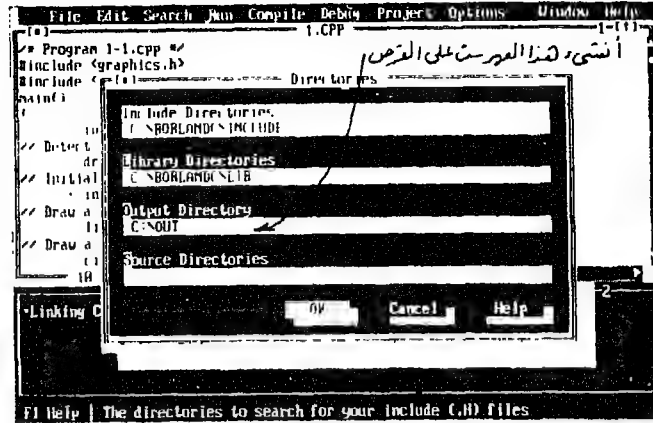
وبطبيعة الحال فإنه بجانب مكتبة الرسم يلزم إعداد الممرات المناسبة بالصورة المعتادة بحيث تشمل الفهرست الفرعي INCLUDE والفهرست الفرعي LIB .

(ويجوز أن يتضمن الممر أية فهرس فرعية أخرى لازمة مثل فهرس المكتبات اللازمة لبرمجة التوافد) .

والشكل التالي يوضح نافذة إعداد الممرات لكل من بورلاند سي أو تيريو سي++ . ونلاحظ وجود الفهرست :

C:\out

والذى سوف نستخدمه لتخزين الملفات التنفيذية للمشروعات وعليك بإنشاء فهرست بهذا الاسم على القرص الصلب "C" . حيث أن برامج المشروعات الموجودة على القرص المصاحب للكتاب تستخدم هذا الفهرست . أو يمكنك إذا شئت تغيير هذا الاسم إلى ما يناسبك مع تعديل اسم الفهرست إلى الاسم الجديد عند فتح أى مشروع من المشروعات الموجودة على القرص .



شكل (٤)

إعداد الممرات للفهارس الفرعية

ويتم الوصول إلى نافذة الممرات للفهارس باستخدام الاختيار :

Options

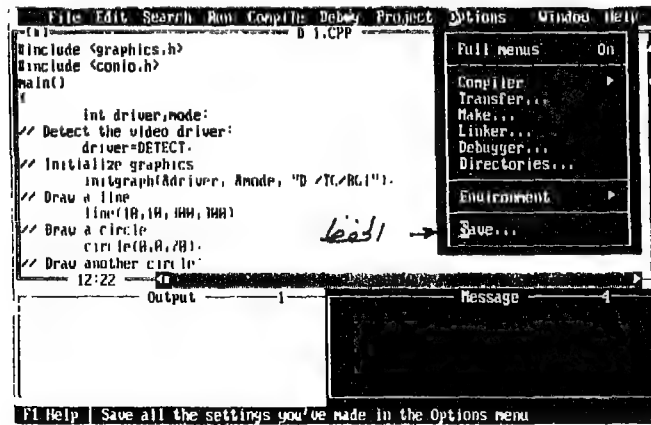
Directories

ثم اختيار كلمة :

وعندما يتم تعديل مواصفات الإعداد للمترجم يمكنك حفظها باستخدام الاختيار :

Save

كما هو موضح بالشكل التالي :



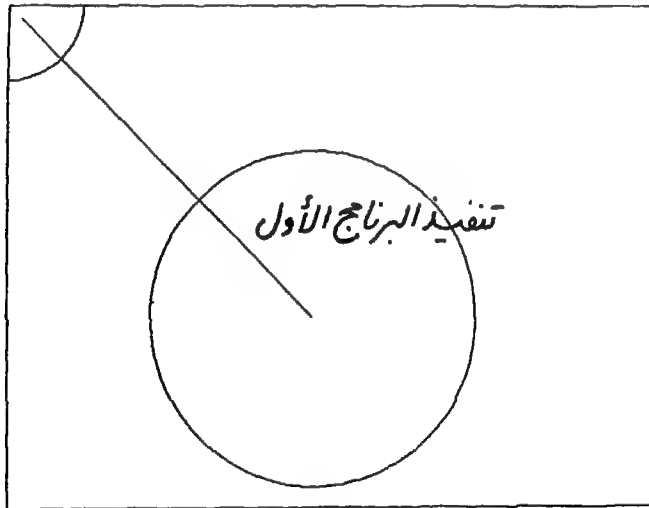
شكل (٥) حفظ مواصفات الإعداد

فلاش

لا يجوز عند استخدام مكتبة الرسم أن نستخدم نموذج الذاكرة الدقيق للمترجم (Tiny Model) .

(١ - ٣) البرنامج الأول

لنبدأ بهذا المثال الذى يرسم لنا على الشاشة مجموعة من الدوائر والخطوط كالموضحة بالشكل التالى :



شكل (٦)

تنفيذ البرنامج الأول

فلنلق نظرة عامة على البرنامج الذى نتج عنه هذا الرسم ثم نلتقى فى مناقشة حول البرنامج .. وهذا هو البرنامج الأول موضحاً فى الشكل التالى :


```

/* Program 1-1.cpp */
#include <graphics.h>
#include <conio.h>
main()
{
    int driver,mode;
    // Detect the video driver:
    driver=DETECT;
    // Initialize graphics:
    initgraph(&driver, &mode, "D:/TC/BGI");
    // Draw a line:
    line(10,10,300,300);
    // Draw a circle:
    circle(0,0,70);
    // Draw another circle:
    circle(300,300,160);
    // Wait for a key:
    getch();
    return (0);
}

```

شكل (٧)

إن أول ما نلاحظه في هذا البرنامج هو استخدام ملف العناوين الخاص بدوال مكتبة الرسم :

graphics.h

وفيما يلي نعلق على عبارات البرنامج المختلفة :

الدخول في نسق الرسم

تبدأ برامج الرسم جميعاً بالتعرف على كارت الفيديو الموجود في الكمبيوتر وضبط طور الرسم (mode) المطلوب . وكما تعلم فإن أطوار الرسم متعددة فهي إما وحيدة اللون أو ملونة وتنقسم الكروت الملونة إلى CGA ، EGA ، VGA ، ولن نهتم في البداية بهذه التفاصيل حيث أن مكتبة الرسم للمترجم قيربوسى تحتوى على إمكانية التعرف الأوتوماتيكي على كارت الرسم وضبط الطور المناسب له .

وكما نرى أن البرنامج يبدأ بإعلان المتغيرين :

- driver وهو يمثل جهاز قيادة الفيديو .
- mode وهو يمثل طور الرسم .

يلي ذلك تخصيص القيمة DETECT للمتغير driver كالآتي :

`driver = DETECT;`

وتعطينا هذه العبارة من الدخول في تفاصيل كروت الرسم في الوقت الحالي حيث يؤدي استخدامها إلى التعرف الأتوماتيكي على كارت الرسم الموجود وإعداد العدة المناسبة للتعامل معه .

ولا يفوتنا هنا أن كلمة DETECT ما هي إلا ثابت مُسمى (named constant) ولا بد من كتابتها بالحروف الكبيرة كما هي . يلي ذلك إعداد الكمبيوتر لاستقبال أوامر الرسم من البرنامج باستخدام الدالة `initgraph` كالآتي :

`initgraph(&driver, &mode, "D:/TC/BGI");`

والدالة `initgraph` تأخذ بارامترات ثلاثة هي :

- مؤشر إلى كارت الرسم `&driver`
- مؤشر إلى طور الرسم `&mode`
- اسم الممر الدال على الفهرست الذي يحتوى ملفات قيادة جهاز الفيديو وهو الفهرست BGI . ونذكرك هنا بضرورة استبدال اسم الممر الوارد في البرنامج باسم الممر المناسب للفهرست BGI (وهو يتفرع دائماً من الفهرست الرئيسي TC للمترجم تيربوسى أو BORLANDC للمترجم بورلاند سى) .

كما يجوز الاستغناء عن اسم الممر والاكتفاء بعلامتى اقتباس فارغتين بشرط أن يكون ملف قيادة الفيديو المناسب موجوداً في الفهرست الحالي . فإذا كنت تستخدم الكارت VGA مثلاً فيلزمك نسخ الملف .

EGAVGA.BGI

إلى الفهرست الحالي . أما إذا كنت تستخدم الكارت CGA فعليك بنسخ الملف :

CGA.BGI

إلى الفهرست الحالي .

كما نلاحظ عند كتابة اسم الممر ظهور الشرط المائلة إلى اليمين بدلاً من الشرط المائلة إلى اليسار . ويجوز مع ذلك استخدام الشرط المائلة إلى اليسار على أن نستخدم شرطين بدلاً من شرط واحد (كما نعلم أن الشرط المائلة إلى اليسار لها معنى خاص في لغة سي) أى :

```
initgraph(&driver, &mode, "D:\\TC\\BGI");
```

والصيغة العامة للدالة `initgraph` تتبع عينة الدالة (prototype) الآتية :

```
void far initgraph(int far *driver,
                  int far *mode,
                  char far *path);
```

شكل (٨)

وبتنفيذ الدالة `initgraph` يصبح الكمبيوتر جاهزاً على استقبال الأوامر من دوال الرسم المختلفة ؛ ويصطلح على أن الكمبيوتر في هذه الحالة قد أصبح في نسق الرسم (graphics mode) .

ويختلف نسق الرسم في خصائصه — كما سنرى — عن نسق الكتابة (text mode) المعتاد .

دالة رسم دائرة circle

كما نرى في البرنامج فإن دالة رسم الدائرة هي الدالة `circle` وهي تأخذ بارامترات ثلاثة هي إحداثيات المركز x,y ونصف القطر r كالمثال الآتي :

circle(300,300,160);

وتؤدي هذه الدالة إلى رسم الدائرة الكبيرة المتمركزة عند النقطة (300,300) والتي نصف قطرها 160 .

ومن الواضح أن الأبعاد في مجال الرسم تقاس بعدد البكسلات (pixels) . ونذكرك بأن البكسل هي أصغر نقطة يمكن إضاءتها على الشاشة وهي مختصر للعبارة "picture cell" بمعنى خلية الصورة .

وتختلف عدد البكسلات على سطح الشاشة باختلاف كارت الرسم وطور الرسم المستخدم . والبرنامج المستخدم هنا تم تنفيذه على كومبيوتر ذى كارت VGA باستخدام طور الرسم ذى الدقة العالية VGAHI الذى يمنحنا درجة الدقة :

$$640 \times 480$$

(سوف يلى الحديث عن أطوار الرسم بالتفصيل) .

معنى هذا أن الشاشة تحتوى على عدد ٦٤٠ بكسل في كل سطر أفقى وتحتوى على ٤٨٠ بكسل في كل عمود رأسى . ويعطى حاصل الضرب ٦٤٠ × ٤٨٠ عدد البكسلات المكونة للصورة على سطح الشاشة .

أما نقطة الأصل — في مجال الرسم — فهي النقطة (0,0) وهى تقع عند الركن الأيسر العلوى للشاشة . وفي برنامجنا الحالى فإن عبارة رسم الدائرة الثانية :

circle(0,0,70);

ترسم دائرة مركزها نقطة الأصل تماماً .

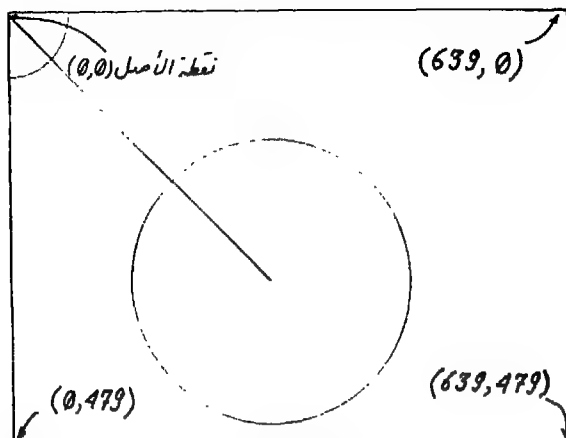
ولأن نقطة الأصل تبدأ من الموقع (0,0) فإن أركان الشاشة تأخذ الإحداثيات الآتية :

(0,0)

(639,0)

(639,479)

(0,479)



شكل (٩)

أما عينة الدالة circle فهي تتبع الصيغة الآتية :

```
void far circle(int x, int y,
               int radius);
```

شكل (١٠)

دالة رسم مستقيم line

تستخدم الدالة line لرسم الخطوط المستقيمة وهي تأخذ أربعة بارامترات تمثل إحداثيات النقطتين اللتين يصل بينهما الخط :

```
line(10,10,300,300);
```

في هذه العبارة يمتد المستقيم ما بين النقطة (10,10) إلى النقطة (300,300) .
انظر إحداثيات الشاشة في الشكل (٩) .

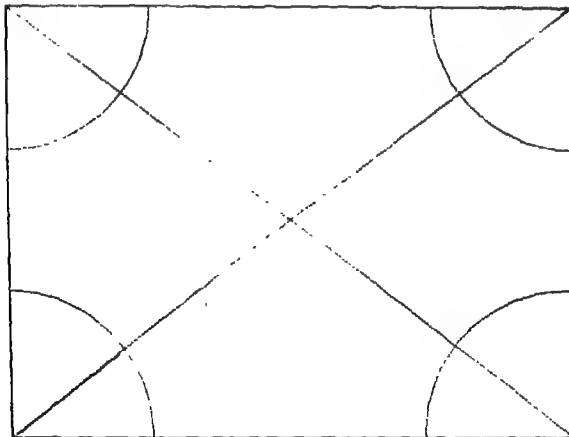
ونأخذ الدالة line الصورة العامة الآتية :

```
void far line(int start_X, int start_Y,
              int end_X, int end_Y);
```

شكل (١١)

تدريب (١ - ١)

أجر التعديلات اللازمة على البرنامج السابق بحيث يؤدي إلى رسم الشكل التالي ، لاحظ أن مركز الدوائر يقع في أركان الشاشة تماماً .
(ملاحظة : البرواز الخارجى للصورة غير مطلوب) .



شكل (١٢)

صور مختلفة لدالة رسم المستقيمات *lineto*, *linere*

تمدنا مكتبة الرسم بلغة تيربوسى (أو بورلاند سى) بصور مختلفة لدالة رسم الخط المستقيم تناسب الأغراض المختلفة ، مثل الدالتين *lineto* ، *linere* .

والدالة **lineto** تستخدم في رسم خط مستقيم من الموقع الحالي إلى نقطة جديدة . ولنفرض أن الموقع الحالي لنقطة الرسم هو (x,y) ؛ فإذا بدأنا برسم خط مستقيم بالعبار **line** ما بين النقطتين (x,y) ، (x_1,y_1) فإن الموقع الحالي يظل هو الإحداثي (x,y) ، بمعنى أن الموقع الحالي لا يتغير كنتيجة لاستخدام العبارة **line** .

لذلك فإن أردنا رسم مستقيم جديد يخرج من نقطة الموقع الحالي (x,y) فلا نحتاج في هذه الحالة إلى تكرار نقطة البداية . يكفي استخدام عبارة مثل :

lineto(200,300);

بموجب هذه العبارة يمتد الخط المستقيم ما بين الموقع الحالي إلى النقطة الجديدة (200,300) .

وتتميز الدالة **lineto** بأنها تؤدي إلى تغيير نقطة الموقع الحالي إلى النقطة الجديدة . فلو أننا رسمنا مستقيماً جديداً بالعبارة :

lineto(250,380);

فسوف يبدأ هذا المستقيم من النقطة السابقة (200,300) والتي تعتبر هي الموقع الحالي الجديد .

وتأخذ عينة الدالة **lineto** الصورة الآتية :

void far lineto(int X, int Y);

شكل (١٣)

أما الدالة **linereel** فهي ترسم خطاً مستقيماً يمتد من الموقع الحالي إلى نقطة جديدة تبعد بمسافة معينة — عن الموقع الحالي — في الاتجاه الأفقي ، وبمسافة معينة — عن الموقع الحالي — في الاتجاه الرأسى . ويؤدي استخدام هذه الدالة إلى تغيير إحداثيات الموقع الحالي إلى الموقع الجديد .

ولذلك فإن الدالة linerel (كما هو واضح من اسمها) تنقلنا دائماً إلى موقع جديد منسوب إلى «الموقع الحالي» الأخير. وتأخذ الدالة linerel الصورة العامة الآتية:

```
void far linerel(int delta_X, int delta_Y);
```

شكل (١٤)

● ● وتجب ملاحظة أنه إذا لم يتم تحديد الموقع الحالي (مع أى دالة من الدوال) فإن الرسم دائماً يبدأ من نقطة الأصل ● ●
ولكى نضرب مثلاً لاستخدام الدالة linerel فلنرسم بروازاً يمثل حدود الشاشة باستخدام البرنامج الآتى:

```
/* Program 1-2.cpp */
#include <graphics.h>
#include <conio.h>
main()
{
    int driver,mode;
    driver=DETECT;
    initgraph(&driver, &mode, "D:/TC/BGI");
    linerel(639,0);
    linerel(0,479);
    linerel(-639,0);
    linerel(0,-479);
    getch();
    return (0);
}
```

شكل (١٥)

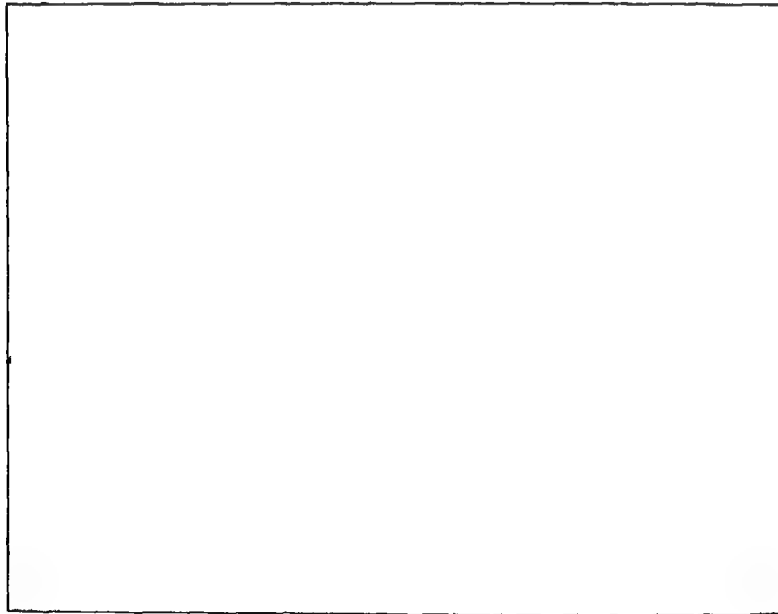
يبدأ هذا البرنامج بمد خط مستقيم من نقطة الأصل إلى نقطة تبعد بمقدار 639 فى الاتجاه الأفقى وبمقدار 0 فى الاتجاه الرأسى ، وهى النقطة الممثلة

للركن الأيمن العلوى للشاشة ، وبهذا تصبح النقطة الأخيرة هى الموقع الحالى للرسم .

يلى ذلك مد خط مستقيم إلى نقطة تبعد بمقدار 479 فى الاتجاه الرأسى وبمقدار 0 فى الاتجاه الأفقى عن الموقع الحالى ، فنصل إلى الركن الأيمن السفلى .

يلى ذلك إزاحة الموقع الحالى بمقدار (639-) أى إلى اليسار فى الاتجاه الأفقى مع الاحتفاظ بالموقع الرأسى (إزاحة 0) ، فنصل إلى الركن الأيسر السفلى .

يلى ذلك إزاحة الموقع إلى أعلى (479-) مع الاحتفاظ بالموقع الأفقى فنصل إلى نقطة الأصل مرة أخرى .



شكل (١٦)

فلاش

يعمل البرنامج الثاني مع كارت الفيديو VGA فإذا استخدمت كارت آخر سوف يخرج الرسم عن حدود الشاشة . يمكنك في هذه الحالة ضبط الأرقام . المحبرة عن أقصى اتصاع وأقصى ارتفاع للشاشة لتناسب كارت الفيديو المستخدم .
(وسوف نتعرض لذلك في الفقرات التالية) .

تدريب (١ - ٢)

نفذ الشكل السابق باستخدام الدالة *lineto* .

(١ - ٤) أطوار الرسم (Graphic modes)

كما ذكرنا من قبل أنه يلزم استخدام الدالة *initgraph* لإعداد الكمبيوتر لاستقبال أوامر الرسم ، ويتضمن ذلك تحميل برامج قيادة الفيديو (video driver) في الذاكرة وتحديد طور الرسم المطلوب .

وفي الفقرات السابقة استخدمنا خاصية التعرف الأتوماتيكي على كارت الفيديو واستخدام طور الرسم سابق التعريف بالاستعانة بالماكرو **. DETECT**

ويمكنك - مع ذلك - تحديد جهاز قيادة الفيديو صراحة ، وكذلك اختيار أحد أطوار الرسم المتاحة للجهاز .

وفي ملف العناوين "graphics.h" قد تم تعريف الثوابت المسماة الآتية التي يمكن استخدامها كأسماء لأجهزة قيادة الفيديو :

MACRO	VALUE
DETECT	0
CGA	1
MCGA	2
EGA	3
EGA64	4
EGAMONO	5
RESERVED	6
HERCMONO	7
ATT400	8
VGA	9
PC3270	10

شكل (١٧) ماكرواات أجهزة القيادة

وعند تحديد جهاز قيادة الفيديو بأية قيمة خلاف القيمة DETECT فإنه يمكنك تحديد طور الرسم باستخدام أحد الثوابت الموضحة في الجدول التالي والمناظرة، لكروت الرسم المختلفة . فعلى سبيل المثال عند اختيار جهاز قيادة الفيديو VGA يجوز لك أن تختار ما بين ثلاثة أطوار للرسم :

- الطور ذو الدقة المنخفضة VGALO وهو يعطى دقة قدرها 640×200 .
- الطور ذو الدقة المتوسطة VGAMED وهو يعطى دقة قدرها 640×350 .
- الطور ذو الدقة العالية VGAHI وهو يعطى دقة قدرها 640×480 .

المواضع	أرقام الأطوار	درجة الدقة	الذوايه	عدد الصفحات	السماء الماكرو للأطوار
CGA	0-3	320 × 200	4	1	CGAC0, CGAC1, CGAC2, CGAC3
	4	640 × 200	2	1	CGAHI
EGA	0	640 × 200	16	4	EGALO
	1	640 × 350	16	2	EGAHI
EGA64	0	640 × 200	16	1	EGA64LO
	1	640 × 350	4	1	EGA64HI
EGAMONO	0	640 × 350	2	1	EGAMONOH
VGA	0	640 × 200	16	2	VGALO
	1	640 × 350	16	2	VGAMED
	2	640 × 480	16	2	VGAHI
MCGA	0-3	320 × 200	4	1	MCGAC0, MCGAC1, MCGAC2, MCGAC3
	4	640 × 200	2	1	MCGAMED
	5	640 × 480	2	1	MCGAHI
HERC	0	720 × 348	2	2	HERCMONOH
ATT400	0-3	320 × 200	4	1	ATT400C0, ATT400C1, ATT400C2, ATT400C3
	4	640 × 200	2	1	ATT400MED
	5	640 × 400	2	1	ATT400HI
PC3270	0	720 × 350	2	1	PC3270HI
IBM8514	0	640 × 480	256	—	IBM8514LO
	1	1024 × 780	256	—	IBM8514HI

شكل (١٨)

أجهزة القيادة وأطوار الرسم المناظرة

(ملاحظة : سوف يلى التعليق على عدد الصفحات فى الباب السادس).

وكما نرى أن لكل جهاز قيادة عدة درجات من الدقة . وعندما نختار كلمة DETECT للتعرف الأوتوماتيكى على كارت الرسم فإن الطور الذى يتم اختياره تلقائياً هو أعلى الأطوار دقة . أما إذا اخترنا اسم جهاز القيادة صراحة مثل CGA أو EGA أو VGA ، فإذا لم نحدد طور الرسم فإن المتغير mode (فى البرنامج 1-1) يأخذ القيمة صفراً وبالتالى يصبح طور الرسم هو الطور المناظر لأقل دقة . فكما نرى بالجدول أن الأطوار ذات الدقة المنخفضة تأخذ القيمة صفراً .

تدريب (١ - ٣)

جرب تخصيص الثوابت الآتية للمتغيرين *driver* ، *mode* في البرنامج "1-1.cpp" وشاهد النتائج . لاحظ أنه لو كان لديك الكارت VGA فيمكنك استخدام كل ما هو دونه (كل الاختيارات) أما لو كان الكارت هو EGA فلا يمكنك اختيار أطوار النظام VGA وهكذا ..

driver = VGA; (١)

mode = VGAHI;

driver = VGA; (٢)

mode = VGALO;

driver = EGA; (٣)

mode = EGAHI;

driver = CGA; (٤)

mode = CGAC0;

driver = CGA; (٥)

mode = CGAC3;

(١ - ٥) استخدام الألوان *Setcolor*

يمكنك تحديد لون الرسم باستخدام الدالة *setcolor* وهي تأخذ الصورة الآتية :

```
void far setcolor(int color);
```

شكل (١٩)

حيث يمثل المتغير color أحد الألوان الموضحة بالجدول التالي وهي عبارة عن ثوابت مسماه (ماكرو).

القيمة	الماكرو في النظام CGA	القيمة	الماكرو في النظام EGA/VGA
0	BLACK	0	EGA_BLACK
1	BLUE	1	EGA_BLUE
2	GREEN	2	EGA_GREEN
3	CYAN	3	EGA_CYAN
4	RED	4	EGA_RED
5	MAGENTA	5	EGA_MAGENTA
6	BROWN	7	EGA_LIGHTGRAY
7	LIGHTGRAY	20	EGA_BROWN
8	DARKGRAY	56	EGA_DARKGRAY
9	LIGHTBLUE	57	EGA_LIGHTBLUE
10	LIGHTGREEN	58	EGA_LIGHTGREEN
11	LIGHTCYAN	59	EGA_LIGHTCYAN
12	LIGHTRED	60	EGA_LIGHTRED
13	LIGHTMAGENTA	61	EGA_LIGHTMAGENTA
14	YELLOW	62	EGA_YELLOW
15	WHITE	63	EGA_WHITE

شكل (٢٠) جدول الألوان

فلنرسم دائرة مثلاً باللون الأصفر عليك أن تسبق عبارة رسم الدائرة بالعبارة الآتية :

```
Setcolor(YELLOW);
```

وهذا يكافئ تماماً استخدام الأرقام المناظرة للماكرو أي :

```
setcolor(14);
```

كما يمكنك التحكم في لون الخلفية أيضاً باستخدام الدالة setbkcolor التي تأخذ الصيغة : .

```
void far setbkcolor(int color);
```

شكل (٢١)

وتستخدم مع هذه الدالة نفس الألوان الموضحة بالجدول السابق . ومن الجدير بالذكر أن تحديد لون الخلفية للرسم يعنى لون الخلفية للشاشة كلها ؛ وليس هذا هو الحال عند تلوين خلفية الحروف فى نسق الكتابة (سيلي شرحه) حيث يمكن تلوين كل حرف وخلفيته على حدة .

وفى البرنامج التالى نضيف بعض الألوان المختلفة إلى الرسم الذى أنشأناه بالبرنامج الأول (1-1.cpp) كما نستخدم اللون الأبيض للخلفية .

ونلاحظ أن العبارة "setcolor" تأتى سابقة لأية عبارة رسم .

```
/* Program 1-3.cpp */
#include <graphics.h>
#include <conio.h>
main()
{
    int driver,mode;
    // Detect the video driver:
    driver=DETECT;
    // Initialize graphics:
    initgraph(&driver, &mode, "D:/TC/BGI");
    // Set background color:
    setbkcolor(WHITE);
    // Set foreground color:
    setcolor(MAGENTA);
    // Draw a line:
    line(10,10,300,300);
    // Set foreground color:
    setcolor(GREEN);
    // Draw a circle:
    circle(0,0,70);
    // Set foreground color:
    setcolor(BLUE);
    // Draw another circle:
    circle(300,300,160);
    // Wait for a key:
    getch();
    return (0);
}
```

شكل (٢٢)

استخدام الألوان مع الكارت CGA (حالة خاصة)

انقرض كارت الفيديو CGA من الأسواق نحو عام ١٩٨٢ بعدما أدى دوره في الرسم والتلوين منذ نشأة الكمبيوتر IBM الملون .

وقد كان الكارت CGA فقيراً في إمكانياته ، رديئاً في أدائه وشاقاً في برمجته ، وكان أقصى ما يمكن أن نحصل عليه منه هو الدقة 320×200 . وأربعة ألوان فقط .

ثم جاء الكارت EGA في عام ١٩٨٤ بدقة تصل إلى 640×350 وعدد من الألوان قدره ١٦ لوناً (يتم اختيارها من فئة الألوان الكلية التي يبلغ عددها ٦٤ لوناً) يمكن عرضها في نفس الوقت .

وقد كانت الذاكرة المخصصة للكارت CGA هي 16KB ولكن الذاكرة التي تم تخصيصها للكارت EGA أصبحت 256KB .

وفي عام ١٩٨٧ ظهر الكارت VGA مواكباً لأجهزة الجيل الثاني من الكمبيوتر آى بي إم (PS/2) وقد بلغت الدقة التي يمكن الحصول عليها من هذا الكارت 640×480 بعدد 256 لوناً .

ويعتبر الكارت VGA هو الكارت الشائع هذه الأيام فهو يؤدي كل ما تؤديه الكروت السابقة له . ومع ذلك فالكارت EGA والشاشات التي تعمل معه لازالت مطروحة في الأسواق بأسعار أقل ، ولكن انخفاض أسعار المعدات بصفة عامة أدت إلى اختفاء الكارت الرائد CGA من الساحة غير مأسوف عليه .

وليس من المتوقع أن يكون قارئ هذا الكتاب يستخدم الكارت CGA في الرسم ومع ذلك فسوف نقدم هذه الفقرة من باب الاحتياط وهي لا تهم إلا مستخدمي الكارت CGA .

فلاش

قامت شركة آى بى إم فى عام ١٩٩٠ الكارت الجديد XGA الذى تصل دقته إلى 1024×768 ولكن لغات سى المختلفة لا تتضمن وسائل برمجة هذا الكارت حتى الآن .

تعمل ألوان النظام CGA من خلال لوحة الألوان (Palette) الموضحة بالجدول التالى :

اللون رقم 3	اللون رقم 2	اللون رقم 1	رقم لوحة الألوان
أصفر	أحمر فاتح	أخضر فاتح	0
أبيض	طوبى فاتح	سماوى فاتح	1
بنى	أحمر	أخضر	2
رمادى فاتح	طوبى	سماوى	3

وكما نرى بالجدول أنه توجد أربعة لوحات للألوان (من صفر إلى 3) . ويتم اختيار لوحات الألوان باختيار أحد الأطوار المناظرة للكارت CGA (والتي سبق تقديمها بالشكل ١٨) . وهى :

CGAC0

CGAC1

CGAC2

CGAC3

وباختيار لوحة الألوان تتحدد الألوان الثلاثة الممكن استخدامها وفقاً للجدول السابق . فإذا اخترنا اللوحة CGAC0 مثلاً ، أصبح فى إمكاننا استخدام الألوان الموجودة بالصف الأول من الجدول . أى الأخضر الفاتح (اللون رقم 1) ، والأحمر الفاتح (اللون رقم 2) ، والأصفر (اللون رقم 3) .

أما اللون الرابع فهو يختص بالخلفية ويجوز تحديده بالطريقة المعتادة باختيار أى لون من مجموعة الألوان ذات الستة عشر لوناً (من الجدول شكل (٢٠)).

● مثال :

● اختيار لوحة الألوان رقم 0 :

```
driver = CGA;
mode = CGAC0;
```

● اختيار اللون الأصفر للرسم :

```
setcolor(3);
```

● اختيار اللون الأحمر الفاتح للرسم :

```
setcolor(2)
```

● اختيار اللون الأرجوانى للخلفية :

```
setbkcolor(MAGENTA);
```

والبرنامج التالى يعرض مثالاً كاملاً لرسم مستقيم ودائرتين باستخدام الكارت CGA ونلاحظ هنا أن الأبعاد المستخدمة فى الرسم أقل من مثيلاتها فى البرنامج الأول لأن حدود الشاشة لا تتعدى 320×200 كما ذكرنا . ومن الجدير بالملاحظة هنا أنك لو استخدمت أرقاماً أكبر من حدود الشاشة فلن يعترض المترجم على ذلك ؛ كل ما يحدث أن الرسم سيخرج عن حدود الشاشة .

ملاحظة

ليس بالضرورة أن يكون لديك كارت الرسم CGA حتى تجرب هذا البرنامج . إن الكروت EGA ، VGA يمكنها تنفيذ الطور CGA كطور متدهور من أطوار الرسم .

```

/* Program 1-4.cpp */
// CGA graphics and colors
#include <graphics.h>
#include <iostream.h>
#include <conio.h>
main()
{
    int driver,mode;
// Choose driver:
    driver=CGA;
// Choose mode (palette):
    mode=CGAC0;
    initgraph(&driver, &mode, "D:/TC/BGI");
// Set background color:
    setbkcolor(MAGENTA);
// Set color:
    setcolor(1);
// Draw a line:
    line(10,10,200,100);
// Set color:
    setcolor(2);
// Draw a circle:
    circle(60,70,70);
// Set color:
    setcolor(3);
// Draw a circle:
    circle(100,100,90);
    getch();
    return (0);
}

```

شكل (٢٣)

(٦ - ١) استكشاف المعدات detectgraph

ربما تعمل على كومبيوتر ما ولا تعلم على وجه اليقين نوع الكارت المستخدم . وربما ترغب في معرفة نوع كارت الرسم ولو من باب حب الاستطلاع . في هذه الحالة عليك بالدالة :

detectgraph

وتستخدم هذه الدالة للحصول على القيم العددية التي تناظر نوع جهاز قيادة الفيديو (driver) وطور الرسم سابق التعريف (mode) المناظر لأعلى دقة .

وعينة هذه الدالة كالآتي :

```
void far detectgraph(int far driver*,
                    int far *mode);
```

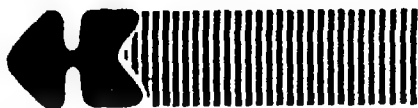
شكل (٢٤)

ولو كان الكمبيوتر لا يحتوى على كارت للرسم فإن بارامتر ملف قيادة الفيديو (driver) يُرجع القيمة (-2) .

وهناك تحفظ على استخدام هذه الدالة مع كارت الرسم IBM 8514 .

وفي المثال التالى نستخدم هذه الدالة لاستكشاف معدات الكمبيوتر الذى نعمل عليه . وكما نرى فى تنفيذ البرنامج أن القيم العددية هى :

- driver=9 وهذا يناظر الكارت VGA (انظر الجدول) .
- mode=2 وهذا يناظر طور الرسم VGAHI (انظر الجدول) .



```

/* Program 1-5.cpp */
#include <graphics.h>
#include <iostream.h>
#include <conio.h>
main()
{
    int driver,mode;
    detectgraph(&driver, &mode);
    cout << endl << "This is your hardware report:"
         << endl << ". Graphic driver #" << driver
         << endl << ". Graphic mode #" << mode
         << endl << "Press any key to continue..";
    getch();
    return (0);
}

```

This is your hardware report:
 . Graphic driver #9
 . Graphic mode #2
 Press any key to continue..

شكل (٢٥)

<i>getgraphmode</i>	استكشاف أطوار الرسم
<i>getmoderange</i>	

أما هذه المجموعة من الدوال فهي تستخدم لمعرفة الطور الحالي للرسم أو لمعرفة الأطوار المتاحة لجهاز القيادة المعين ، والمعلومة الأخيرة هي نفسها المعلومة التي نحصل عليها من فحص الجدول شكل (١٨) .
 والدالة الأولى هي :

getgraphmode

وهي تستطلع طور الرسم الحالي وترجع قيمته . وبطبيعة الحال لابد من الدخول في نسق الرسم بالدالة **initgraph** قبل استخدام هذه الدالة . وعينة الدالة كالآتي :

```
int far getgraphmode(void);
```

شكل (٢٦)

أما الدالة الثانية فهي :

```
getmoderange
```

وعينة هذه الدالة كالآتي :

```
void far getmoderange(int driver,
                      int far *lowmode,
                      int far *himode);
```

شكل (٢٧)

وكما نرى من عينة الدالة أنها تأخذ البارامترات الثلاثة الآتية :

- driver ويمثل ملف قيادة جهاز الفيديو . ويلزم إمداد الدالة بهذا البارامتر .
 - lowmode الطور المناظر لأقل دقة لجهاز الفيديو المقصود .
 - himode الطور المناظر لأعلى دقة لجهاز الفيديو المقصود .
- والمثال الآتي يوضح كيفية استخدام الدالة للحصول على معلومات جهاز قيادة الفيديو CGA:

```
getmoderange(CGA, &lowmode, &himode);
```

بعد هذا الاستدعاء فإن المتغير lowmode يعطى القيمة صفرًا بينما يعطى المتغير himode القيمة 4 (راجع الجدول) .

تغيير طور الرسم *setgraphmode*

أما هذه الدالة فهي تستخدم لتغيير طور الرسم الحالى إلى طور جديد وتأخذ الدالة الصيغة الآتية :

```
void far setgraphmode(int mode);
```

شكل (٢٨)

وتأخذ الدالة بارامتراً واحداً وهو طور الرسم المطلوب الانتقال إليه . ومن البديهي أنه لا يجوز استخدام هذه الدالة ما لم ندخل إلى نسق الرسم مبدئياً بالدالة *initgraph* .

والبرنامج التالى يوضح استخدام الدوال الثلاث الأخيرة فى استكشاف أطوار الرسم لجهاز الفيديو المستخدم ، ثم تغيير الطور الحالى للرسم إلى الطور *VGALO* ، وفى النهاية نستخدم الدالة *getgraphmode* للتحقق من الطور الجديد وطباعته على الشاشة .



```

/* Program 1-6.cpp */
#include <graphics.h>
#include <iostream.h>
#include <conio.h>
main()
{
    int driver, mode;
    int himode, lowmode;
    // Detect the video driver:
    driver=DETECT;
    // Initialize graphics:
    initgraph(&driver, &mode, "D:/TC/BGI");
    // Display the available mode ranges:
    getmoderange(driver, &lowmode, &himode);
    cout << endl
         << "This is your hardware report:"
         << endl
         << ". Graphic driver #" << driver
         << endl
         << ". Graphic lowest mode #" << lowmode
         << endl
         << ". Graphic highest mode #" << himode
         << endl
         << ". Current graphic mode is #" << mode
         << endl
         << "Press any key to continue..";
    getch();
    // Change and print mode:
    setgraphmode(VGALO);
    cout << endl
         << "Now the graphic mode is #"
         << getgraphmode()
         << endl
         << "Press any key to continue..";
    getch();
    return (0);
}

```

شكل (٢٩)

البرنامج السادس

عند تشغيل هذا البرنامج فإنه يطبع على الشاشة التقرير الآتي :


```
This is your hardware report:
. Graphic driver #9
. Graphic lowest mode #0
. Graphic highest mode #2
. Current graphic mode is #2
Press any key to continue..
Now the graphic mode is #0
Press any key to continue..
```

شكل (٣٠) تنفيذ البرنامج السادس

وكما نرى أن البرنامج يبدأ بالدخول في نسق الرسم بعد التعرف على كارت الرسم الموجود بالكمبيوتر وضبط طور الرسم إلى أعلى دقة وبذلك يصبح المتغيران "driver"، "mode" معلومي القيمة .

يلي ذلك استخدام الدالة `getmoderange` لإيجاد خصائص ملف قيادة الفيديو "driver" فنحصل على أطوار الرسم الموضحة بالشكل (٣٠) .

ويتم بعد ذلك تغيير طور الرسم إلى الطور `VGA0` بالدالة `setgraphmode` والتحقق منه وطباعته باستخدام الدالة `getgraphmode` .

ومن الجدير بالذكر أن الحروف المكتوبة على الشاشة سوف تظهر مختلفة عما في الشكل السابق حيث نراها كما هي مطبوعة على الورق . ففي الجزء الأول من البرنامج ، عندما تكون الشاشة في الطور `VGAHI` تظهر الكتاب أصغر حجماً وأكثر دقة ؛ أما في الجزء الثاني عندما تتحول الشاشة إلى الطور `VGA0` تبدو الحروف أكبر وتصبح الدقة أقل .

وفي كلتا الحالتين فإن الكتابة التي تظهر على الشاشة في نسق الرسم تختلف عن الكتابة المعتادة التي تظهر في نسق الكتابة . وأهم ما يميز نسق الرسم عن نسق الكتابة أننا لا نرى نقطة الكتابة (cursor) التي تحفك على الشاشة في الواضع المعتاد .

ولنا في الباب القادم جولة جديدة موسعة مع الكتابة في نسق الرسم والكتابة .

الموجز

[١] تعرفنا في هذا الباب على نسق جديد بخلاف نسق الكتابة (text mode) المعتاد وهو نسق الرسم (graphic mode) وعرفنا انه لدخول هذا النسق يلزم إمداد البرنامج بجهاز قيادة الفيديو (video driver) وبطور الرسم المطلوب استخدامه (video mode) ، وكذلك بالفهرست الذى يحتوى على أجهزة قيادة الفيديو المختلفة .

وقد تعرفنا بدالة إعداد نسق الرسم ، وعرفنا ان اطوار الرسم وأجهزة قيادة الفيديو مشفرة فى صورة ثوابت مَسَمَاة او ماكرو مثل VGAMED, VGA .

[٢] تعرفنا فى هذا الباب بالدوال الآتية :

دالة إعداد نسق الرسم

```
void far initgraph(int far *driver,
                   int far *mode,
                   char far *path);
```

دالة رسم خط مستقيم ما بين نقطتين

```
void far line(int start_X, int start_Y,
              int end_X, int end_Y);
```

دالة رسم مستقيم من النقطة الحالية إلى نقطة معلومة

```
void far lineto(int X, int Y);
```

دالة رسم مستقيم من النقطة الحالية إلى نقطة ذات إزاحة نسبية

```
void far linerel(int delta_X, int delta_Y);
```

دالة رسم دائرة ذات مركز معلوم ونصف قطر معلوم

```
void far circle(int x, int y,
               int radius);
```

دالة تغيير لون الواجهة (الرسم)

```
void far setcolor(int color);
```

دالة تغيير لون الخلفية

```
void far setbkcolor(int color);
```

دالة استكشاف المعدات

```
void far detectgraph(int far driver*,
                    int far *mode);
```

دالة استكشاف طور الرسم

```
int far getgraphmode(void);
```

دالة تغيير طور الرسم

```
void far setgraphmode(int mode);
```

دالة استكشاف اطوار الرسم لجهاز معين للفيديو

```
void far getmoderange(int driver,
                    int far *lowmode,
                    int far *himode);
```

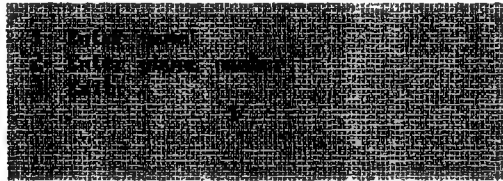
وجميع هذه الدوال تم تعريفها بالملف "graphics.h".

[٣] باستخدام هذه المجموعة من الدوال عرفنا كيفية التعامل مع معدات الرسم، كما استخدمنا دوال الرسم في بعض التطبيقات لرسم الأشكال البسيطة والتعرف على أبعاد الشاشة.

الباب الثانى

العمل فى نسق الكتابة

Text Screen



Sam A. Abolrous

651-6052

مفتّح

إن نسق الكتابة (Text mode) يختص ببرمجة النصوص باستخدام اللبّات أسكى المعروفة . والشاشة فى هذا النسق تتكون من سطور وأعمدة كما هو الحال فى بيئة نظام التشغيل . ومع ذلك فإن لغة سى تمدنا فى هذا النسق بإمكانات قوية لبرمجة النوافذ والألوان والتحكم فى موقع مؤشر الكتابة .

وهذه هى معظم الإمكانيات التى يطلبها المبرمج عندما ينشئ فى برنامج الوصلة البينية التى تمثل شاشات إدخال البيانات أو طباعة النتائج .

(٢ - ١) نسق الرسم ونسق الكتابة

Text mode and graphics mode

تنقسم الشاشة فى نسق الكتابة إلى صفوف وأعمدة . وقد يكون عدد الأعمدة 40 أو 80 عموداً ، وتنقسم الشاشة لعدد 25 أو 43 أو 50 صفاً . وتظهر اللينات على الشاشة فى مساحات ثابتة عند تقاطع الصفوف والأعمدة فى صورة لبنات آسكى ASCII (انظر الملحق ب) وتتميز كل لبنة بمجموعة من الصفات مثل اللون وشدة الإضاءة إلى آخره .

ونعدنا لغة مى (و مى++) بمجموعة من الأدوات لعرض اللينات على الشاشة فى بيئة الكتابة وكذلك للتحكم فى صفاتها المختلفة .

أما فى بيئة الرسم فإن الشاشة — كما ذكرنا من قبل — تنقسم إلى خلايا صغيرة تسمى البكسلات (pixels) ويعتمد عدد البكسلات فى صفحة الشاشة على كارت الرسم وعلى طور الرسم المستخدم .

وفى بيئة الرسم فإن إحداثى نقطة الأصل التى تقع فى الركن الأيسر العلوى هو (0,0) . أما فى بيئة الكتابة فإن الركن الأيسر العلوى يناظر الإحداثى (1,1) .

مفهوم النافذة

يمكنك ، سواء فى بيئة الرسم أو بيئة الكتابة ، أن تحدد المشهد خلال مستطيل ذى أبعاد معينة بحيث أن كل ما تكتبه أو ترسمه يظهر بداخل حدود النافذة بدون أن يؤثر على بقية محتويات الشاشة .

وقد اصطلح فى لغة تيربوسى على تسمية النافذة فى بيئة النصوص بالاسم

WINDOW أما في بيئة الرسم فقد اصطلح عليها بالاسم **VIEWPORT** لكنهما من حيث المضمون يعنيان نفس الشيء .

وعندما نتحدث عن نقطة الأصل في مجال النوافذ فإننا دائماً نعني الركن الأيسر العلوي للنافذة . ولذلك فإن لكل نافذة نقطة أصل تمثل الإحداثي (0,0) في بيئة الرسم أو الإحداثي (1,1) في بيئة الكتابة .

والنافذة قد تكون مستطيلةً صغيراً وقد تمتد لتشمل الشاشة كلها وهذا هو الوضع سابق التعريف .

أما عن دوال لغة سى المستخدمة سواء في الرسم أو الكتابة فهي تتعامل مع الإحداثيات النسبية للنوافذ كما سنرى (فيما عدا الدوال التي تستخدم في خلق النوافذ وتعريفها) .

الخروج من نسق الرسم إلى نسق الكتابة

في البرنامج الأخير بالباب الأول استخدمنا دوال الطباعة القياسية في طباعة بعض النصوص على شاشة الرسم ، وكما رأينا أن الكتابة على شاشة الرسم كانت مختلف تماماً عن الكتابة المعتادة التي نشاهدها في نسق الكتابة ولا سيما عندما استخدمنا طور الرسم ذا الدقة المنخفضة . والشاشة تظل على هذا الحال ما لم نستخدم الدالة المناسبة التي تعيد الشاشة إلى حالتها الأولى . هذه الدالة هي :

closegraph

تؤدي هذه الدالة إلى إنهاء نسق الرسم والعودة إلى نسق الكتابة ولذلك فإن أى عبارة طباعة تالية سوف تؤدي إلى ظهور النص بالطريقة المعتادة على الشاشة .

وعينة هذه الدالة كالآتي :


```
#include <graphics.h>

void far closegraph(void);
```

شكل (١)

ويؤدي استخدام هذه الدالة أيضاً إلى تحرير الحيز من الذاكرة الذي كان مستخدماً بواسطة بيئة الرسم لتخزين ملفات قيادة الأجهزة وسائر مستلزمات الرسم . وهي تستخدم عادة عند الجمع ما بين نسقي الرسم والكتابة . وفي البرنامج التالي قد أضفنا هذه الدالة إلى مؤخرة البرنامج "6.cpp-1" تلها عبارة لطباعة النص :

"Hey, I am not in graphics any more!".

وفي مجال الطباعة على الشاشة قد استخدمنا هنا دالة جديدة تؤدي إلى طباعة النص في إحداثي معين على الشاشة وهي الدالة :

gotoxy

وكما نرى أن اسم الدالة يعبر عن معناها فهي تقول "اذهب إلى الإحداثي x,y ، حيث "x" هو رقم العمود و "y" هو رقم الصف . وتأخذ عينة الدالة الصورة الآتية :

```
#include <conio.h>

void gotoxy(int x, int y);
```

شكل (٢)

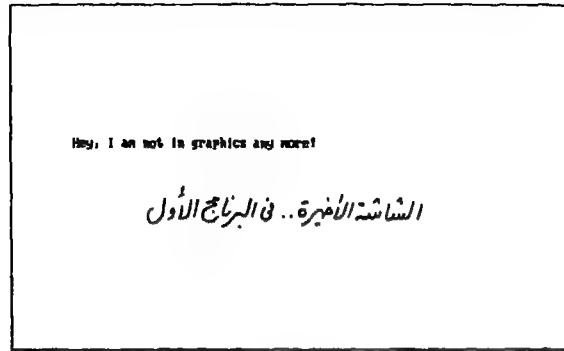
```

/* Program 2-1.cpp */
#include <graphics.h>
#include <iostream.h>
#include <conio.h>
main()
{
    int driver, mode;
    int himode, lowmode;
    // Detect the video driver:
    driver=DETECT;
    // Initialize graphics:
    initgraph(&driver, &mode, "D:/TC/BGI");
    // Display the available mode ranges:
    getmoderange(driver, &lowmode, &himode);
    cout << endl
         << "This is your hardware report:"
         << endl
         << ". Graphic driver #" << driver
         << endl
         << ". Graphic lowest mode #" << lowmode
         << endl
         << ". Graphic highest mode #" << himode
         << endl
         << ". Current graphic mode is #" << mode
         << endl
         << "Press any key to continue..";
    getch();
    // Change and print mode:
    setgraphmode(VGALO);
    cout << endl
         << "Now the graphic mode is #"
         << getgraphmode()
         << endl
         << "Press any key to continue..";
    getch();
    // Back to text mode:
    closegraph();
    gotoxy(10,10);
    cout << "Hey, I am not in graphics any more!";
    getch();
    return (0);
}

```

شكل (٣)
البرنامج الأول





شكل (٤)

الجمع بين الرسم والكتابة في نسق الرسم

يمكنك إظهار النصوص والرسم معاً في شاشة واحدة باستخدام عبارات الطباعة المعتادة ؛ ويجوز استخدام الدالة "gotoxy" لطباعة العبارات في موضع معين على الشاشة . ومع ذلك فهناك إمكانات خاصة لمعالجة النصوص في بيئة الرسم (وكذلك في بيئة الكتابة) سوف نعرضها تباعاً .

وفي البرنامج التالي قد أضفنا بعض العبارات إلى الرسم الأول الذي رسمناه بالبرنامج "1-1.cpp" . وقد استخدمنا هنا الدالة cout للطباعة وقد كان من الجائز استخدام الدالة التقليدية printf .



```

/* Program 2-2.cpp */
#include <graphics.h>
#include <iostream.h>
#include <conio.h>
main()
{
    int driver,mode;
    driver=DETECT;
    initgraph(&driver, &mode, "D:/TC/BGI");
    // Set color:
    setcolor(GREEN);
    // Draw a line:
    line(10,10,300,300);
    // Set color:
    setcolor(CYAN);
    // Draw a circle:
    circle(0,0,70);
    // Set color:
    setcolor(RED);
    // Draw a circle:
    circle(300,300,160);
    // Set the sursor position:
    gotoxy(10,13);
    cout << "HI THERE!.. These are some of C graphics..";
    // Set the sursor position:
    gotoxy(40,20);
    cout << "Ossama Al-Husseiny";
    getch();
    return (0);
}

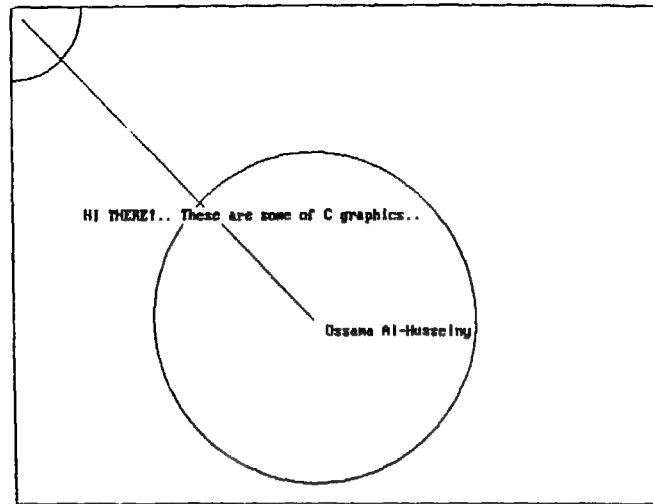
```

شكل (٥)

البرنامج الثاني

والشكل التالي يوضح المشهد الذي يظهر على الشاشة عند تنفيذ البرنامج .





شكل (٦)

ونلاحظ في البرنامج أن عبارة وضع مؤشر الكتابة gotoxy تأتي دائماً
سابقة لعبارة الطباعة مثل :

gotoxy(10,13)

وتجب ملاحظة أنه لو احتوت عبارة الطباعة على علامة نهاية السطر
"\\n" أو على كلمة endl فإن مؤشر الكتابة يعود إلى أول السطر عند هذه
النقطة .

(٢ - ٢) الأطوار في نسق الكتابة

(Text Modes)

يمكنك الانتقال إلى أحد أطوار نسق الكتابة باستخدام الدالة :

textmode

وهي تمكنك من الدخول في أحد أطوار ستة من أطوار الكتابة وهي جميعاً
موضحة بالجدول التالي :

القيمة العددية	الماكرو	طور الكتابة (mode)
0	BW40	شاشة ٤٠ عمود - أبيض وأسود
1	C40	شاشة ٤٠ عمود - بالألوان
2	B80	شاشة ٨٠ عمود - أبيض وأسود
3	C80	شاشة ٨٠ عمود - بالألوان
7	MONO	شاشة ٨٠ عمود - لون واحد (monochrome)
- 1	LASTMODE	الطور السابق

شكل (٧)

أطوار نسق الكتابة

وبالطبع فإن استخدام هذه الأطوار يعتمد على نوع الشاشة وكارت الرسم .

وعيقة الدالة textmode تأخذ الصورة الآتية :

```
#include <conio.h>
```

```
void textmode(int mode);
```

شكل (٨)

حيث mode هو أحد الثوابت المسماة (الماكرو) الموضحة بالجدول السابق ، ويجوز بالطبع استخدام القيم العددية المناظرة .

(٢ - ٣) طباعة النصوص في نسق الكتابة

`cprintf`

`cputs`

بالرغم من أنه يمكن استخدام دوال الطباعة المعتادة مثل `puts` و `printf` ولكن بيئة الكتابة تختص ببعض الدوال المشابهة لهذه الدوال والتي تتميز عنها بإمكانية استخدامها مع النوافذ جنباً إلى جنب مع الألوان وسائر إمكانيات بيئة الكتابة . والدوال المستخدمة في نسق الكتابة هي :

`cprintf`

`cputs`

وتستخدم الدالتان في طباعة الحرفيات على الشاشة (أو في النافذة) وتستخدم الدالة الأولى `cprintf` في الطباعة ذات الصيغة (formatted output) . أما الثانية `cputs` فهي ماثلة للدالة `puts` من حيث أنها تستخدم في طباعة الحرفيات بدون صياغة .

وهذه هي عينة الدالة `cprintf` :

```
#include <conio.h>
```

```
int cprintf(const char *format [,argument,...]);
```

شكل (٩)

وكما نرى أنها ماثلة للدالة `printf` في الصيغة لكنها توجه الخرج إلى النافذة كما سنرى في الفقرات التالية .

وهذه هي عينة الدالة `cputs` :

```
#include <conio.h>
```

```
int cputs(const char *str);
```

شكل (١٠)

وهذه الدالة مماثلة للدالة puts فيما عدا أن الخرج يتم توجيهه إلى النافذة. وهناك فارق أساسي بين أداء الدوال cputs ، cprintf والدوال القياسية puts ، printf وهو أن علامة السطر الجديد “\n” لا تجعل الكتابة تبدأ من أول السطر أى أنها لا تترجم إلى “\n\r” كما هو الحال مع الدوال القياسية . فإذا أردت أن تبدأ الكتابة من أول السطر عليك باستخدام اللبئات “\n\r” .

مفكرة

- اللبئة “\n” تناظر الكود أسكى 10 (الكود LF) .
 - اللبئة “\r” تناظر الكود أسكى 13 (الكود CR) .
- (انظر الملحق ب)

(٢ - ٤) استخدام الألوان فى نسق الكتابة

textcolor

تستخدم الدالة textcolor لتلوين الكتابة فى نسق الكتابة وهى تأخذ الصيغة الآتية :


```
#include <conio.h>
```

```
void textcolor(int newcolor);
```

شكل (١١)

أما الألوان المستخدمة في هذه الدالة فقد تكون أحد الألوان الموضحة بالجدول التالي وهي جميعاً مُعرّفة في صورة ماكرو بحيث يسهل استخدامها وتذكرها .

القيمة العددية	الماكرو	اللون
0	BLACK	أسود
1	BLUE	أزرق
2	GREEN	أخضر
3	CYAN	تيركواز
4	RED	أحمر
5	MAGENTA	أرجواني
6	BROWN	بنى
7	LIGHTGRAY	رماسي فاتح
8	DARKGRAY	رماسي غامق
9	LIGHTBLUE	أزرق فاتح
10	LIGHTGREEN	أخضر فاتح

اللون	الماكرو	القيمة العددية
تيركواز فاتح	LIGHTCYAN	11
أحمر فاتح	LIGHTRED	12
أرجواني فاتح	LIGHTMAGENTA	13
أصفر	YELLOW	14
أبيض	WHITE	15
خافق	BLINK	128

شكل (١٢) الوان الواجبة للكتابة

وفي البرنامج التالي نضع هذه الأدوات معاً حيث نبدأ بدخول طور الكتابة C40 الذى يسمح باستخدام الألوان فى شاشة (أو نافذة) تتسع لعدد ٤٠ لينة فى السطر الواحد ، ثم نطبع بعض العبارات الملونة .
ولنلاحظ فى هذا البرنامج استخدام اللينات "١٣١٣" عند الانتقال إلى أول السطر ، وكذلك استخدام اللينة "١٣" للانتقال إلى سطر جديد بدون البدء من أول السطر .



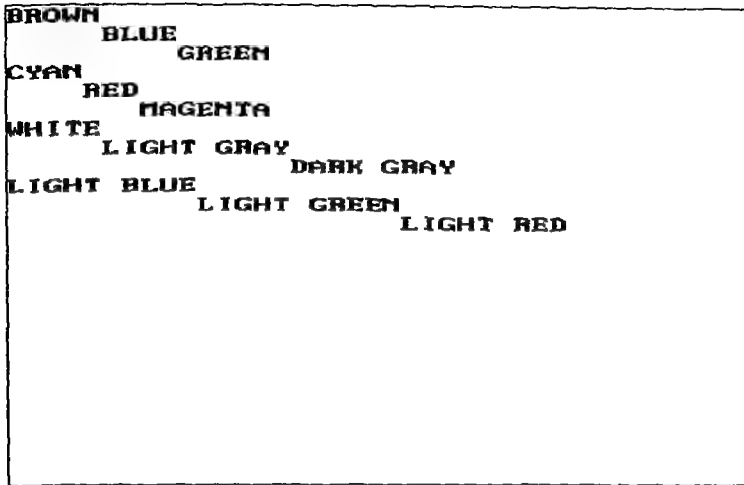
```

/* Program 2-3.cpp */
#include <conio.h>
main()
{
    char *a="BROWN\n";
    char *b="BLUE\n";
    char *c="GREEN\n\r";
    char *d="CYAN\n";
    char *e="RED\n";
    char *f="MAGENTA\n\r";
    char *g="WHITE\n";
    char *h="LIGHT GRAY\n";
    char *i="DARK GRAY\n\r";
    char *j="LIGHT BLUE\n";
    char *k="LIGHT GREEN\n";
    char *l="LIGHT RED\n";
    // Set text mode:
    textmode(C40);
    // Set colors and display output:
    textcolor(BROWN);      cputs(a);
    textcolor(BLUE);       cputs(b);
    textcolor(GREEN);      cputs(c);
    textcolor(CYAN);       cputs(d);
    textcolor(RED);        cputs(e);
    textcolor(MAGENTA);    cputs(f);
    textcolor(WHITE);      cputs(g);
    textcolor(LIGHTGRAY);  cputs(h);
    textcolor(DARKGRAY);   cputs(i);
    textcolor(LIGHTBLUE);  cputs(j);
    textcolor(LIGHTGREEN); cputs(k);
    textcolor(LIGHTRED);   cputs(l);
    getch();
    return (0);
}

```

شكل (١٣)
البرنامج الثالث

وعند تنفيذ هذا البرنامج فإنه يطبع على الشاشة اسم كل لون من الألوان مع إظهار اللون نفسه . وهذا هو التنفيذ بالشكل التالي :



شكل (١٤)

(٢ - ٥) تلوين خلفية الكتابة *textbackground*

يمكنك — علاوة على تلوين النص نفسه — تلوين الخلفية باستخدام الدالة *textbackground* التي تأخذ الصيغة الآتية :

```
#include <conio.h>

void textbackground(int newcolor);
```

شكل (١٥)

والألوان المسموح بها لخلفية الكتابة موضحة بالجدول التالي وأمام كل لون الماكرو المناظر والقيمة العددية المناظرة .

القيمة العددية	الماكرو	اللون
0	BLACK	أسود
1	BLUE	أزرق
2	GREEN	أخضر
3	CYAN	تيركواز
4	RED	أحمر
5	MAGENTA	أرجواني
6	BROWN	بنى

شكل (١٦)
الوان الخلفية للكتابة

وفي البرنامج التالى قد أضفنا تعديلاً جديداً على البرنامج السابق حيث نطبع كل سطر على خلفية لونية مختلفة .



```

/* Program 2-4.cpp */
#include <conio.h>
main()
{
    char *a="WHITE on BLUE background\n\r\n\r";
    char *b="BLACK on MAGENTA background\n\r\n\r";
    char *c="DARK GRAY on CYAN background\n\r\n\r";
    char *d="CYAN on BLACK background\n\r\n\r";
    char *e="RED on GREEN background";
    // Set text mode:
    textmode(C40);
    // Set colors and display output:
    textcolor(WHITE);
        textbackground(BLUE);
            cputs(a);
    textcolor(BLACK);
        textbackground(MAGENTA);
            cputs(b);
    textcolor(DARKGRAY);
        textbackground(CYAN);
            cputs(c);
    textcolor(CYAN);
        textbackground(BLACK);
            cputs(d);
    textcolor(RED);
        textbackground(GREEN);
            cputs(e);

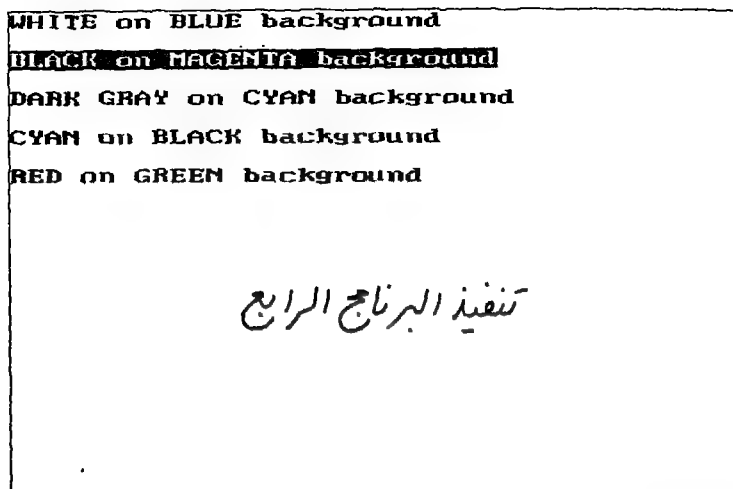
    getch();
    return (0);
}

```

شكل (١٧)

البرنامج الرابع

وفي الشكل التالي نوضح تنفيذ البرنامج مع مراعاة أن درجات الألوان لا تظهر في الطباعة .



شكل (١٨)

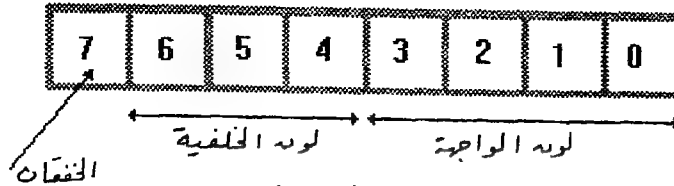
فلاش

جرب استبدال دالة الطباعة `puts` بأحد الدوال القياسية `puts` أو `printf` وسوف تجد أن الألوان تختفى من الشاشة تماماً .

ولا يفوتنا أن نلاحظ أن عبارتي تغيير لون الحروف أو لون الخلفية لا تؤثر على الكتابة الموجودة أصلاً على الشاشة ، بل يسرى تأثيرها على ما يتبعها من عبارات . ولا يتغير لون الواجهة أو لون خلفية حتى يتم تغييره بعبارة جديدة .

(٢ - ٦) تحديد لون الواجهة والخلفية معاً
`textattr`

إن صفات اللبنة (لون الواجهة والخلفية والخفقان) تختزن جميعاً في « بايت » واحدة . كالموضحة بالشكل التالي .



شكل (١٩)

"بايت" الصفات للحروف

ويمكن ضبط جميع صفات الكتابة باستخدام الدالة "textattr" التي يتم بواسطتها التحكم في القيمة العددية الممثلة للصفات مباشرة . وهي تأخذ الصيغة الآتية :

```
#include <conio.h>
```

```
void textattr(int newattr);
```

↑ القيمة العددية للصفات الجديدة

شكل (٢٠)

ويمثل بارامتر الدالة القيمة العددية المخزنة في بايت الصفات وهي قيمة مشفرة بصورة معينة . وطريقة تشفير الألوان يجوز أن تتم كالاتي :

- اضرب لون الخلفية المطلوب (الرقم أو الماكرو) في 16 .
- أجر عملية الجمع المنطقي على لون الواجهة ولون الخلفية .
- إذا أردت إضافة صفة الخفقان للون فأجر عملية الجمع المنطقي للتعبير السابق مع العدد 128 .

ومن البديهي أن مؤثر الجمع المنطقي المستخدم هنا هو مؤثر الجمع المنطقي على مستوى البت "أ" .

◎ مثال :

للحصول على كتابة باللون الأزرق (BLUE) على خلفية بيضاء (WHITE) مع إضافة الخفقان تستخدم العبارة الآتية :


```
textattr(128 | 16*WHITE | BLUE);
```

شكل (٢١) مثال لتحديد صفات الكتابة

والبرنامج التالي يؤدي إلى نفس الألوان التي حصلنا عليها من البرنامج السابق ، ولكننا هذه المرة استخدمنا الدالة `textattr` لتحديد الألوان للعبارة المطبوعة على الشاشة كما أضفنا خاصية الخفقان إلى العبارة الأولى والرابعة . وقد تعمدنا أن نكتب العبارات المختلفة لصفات اللون بطرق غير متطابقة لتوضيح فكرة استخدام مؤثر الجمع المنطقي ، حيث أنه لا قيمة للترتيب الذي تكتب به الصفات .

```
/* Program 2-5.cpp */
#include <conio.h>
main()
{
    char *a="WHITE on BLUE background\n\r\n\r";
    char *b="BLACK on MAGENTA background\n\r\n\r";
    char *c="DARK GRAY on CYAN background\n\r\n\r";
    char *d="CYAN on BLACK background\n\r\n\r";
    char *e="RED on GREEN background";
    // Set text mode:
    textmode(C40);
    // Set colors and display output:

    textattr(128 | WHITE | BLUE*16);
    cputs(a);
    textattr(BLACK | MAGENTA*16);
    cputs(b);
    textattr(DARKGRAY | CYAN*16);
    cputs(c);
    textattr(CYAN | BLACK*16 | 128);
    cputs(d);
    textattr(RED | GREEN*16);
    cputs(e);
    getch();
    return (0);
}
```

شكل (٢٢)

البرامج الخامس

(٢ - ٧) تغيير درجة الإضاءة

highvideo lowvideo normvideo

هناك مجموعة من الدوال تختص بالتحكم فى شدة إضاءة الحروف من خلال ثلاث درجات للإضاءة :

١ - درجة إضاءة عالية :

ويتم الوصول إليها باستخدام الدالة **highvideo** .

٢ - درجة إضاءة خافتة :

ويتم الوصول إليها باستخدام الدالة **lowvideo** .

٣ - درجة الإضاءة المعتادة (سابقة التعريف) :

ويتم الوصول إليها باستخدام الدالة **normvideo**

والآتى بعد عينات الدوال الثلاثة :

```
#include <conio.h>

void highvideo(void);

void lowvideo(void);

void normvideo(void);
```

شكل (٢٣)

واستخدام هذه العبارات مع الألوان يفيد فى تخفيض شدة إضاءة بعض الألوان الفاقعة أو زيادة شدة بعض الألوان الباهتة ، كما أن استخدام درجة الإضاءة المعتادة يلغى اللون الحالى ويظهر الكتابة باللون المستخدم مع نظام

التشغيل (عادة هو اللون الأبيض على خلفية سوداء — ما لم يتم تغييره بوسيلة ما) .

وفي البرنامج التالي نعرض استخدام هذه الدوال الثلاثة مع استخدام الألوان أرقام من ١ إلى ٦ خلال حلقة تكرارية لتغيير اللون بدءاً بالأزرق (رقم 1) إلى البني (رقم 6) .

```
/* Program 2-6.cpp */
#include <conio.h>
main()
{
    int color;
    char *a="\n\r\n\rThis is the low video mode.";
    char *b="\n\rThis is the high video mode.";
    char *c="\n\rThis is the normal video mode.";
    // Set text mode:
    textmode(C80);
    // Set intensity and display output:
    for (color=1;color<=6;color++) {
        textcolor(color);
        lowvideo();
        cputs(a);
        highvideo();
        cputs(b);
        normvideo();
        cputs(c);
    }
    getch();
    return (0);
}
```

شكل (٢٤)

البرنامج السادس

ونلاحظ في البرنامج أن أيّاً من الدوال الثلاثة تأتي بعد تحديد اللون بالدالة

textcolor

```

This is the low video mode.
This is the high video mode.
This is the normal video mode.

This is the low video mode.
This is the high video mode.
This is the normal video mode.

This is the low video mode.
This is the high video mode.
This is the normal video mode.

This is the low video mode.
This is the high video mode.
This is the normal video mode.

This is the low video mode.
This is the high video mode.
This is the normal video mode.

This is the low video mode.
This is the high video mode.
This is the normal video mode.

```

شكل (٢٥)

(٢-٨) دوال إدخال وطباعة اللبنة getch في نسق الكتابة puch

تستخدم الدالة getch لاستقبال لبنة واحدة من لوحة الأزرار في نسق الكتابة ، كما تستخدم الدالة puch لكتابة لبنة واحدة في النافذة . وتوجد عينات هذه الدوال بالملف "conio.h" وهي كالآتي :

```

#include <conio.h>
int getch(void);

#include <conio.h>
int puch(int ch);

```

شكل (٢٦)

وتتميز الدالة puch بأنها لا يتم معها ترجمة اللبنة "\n" إلى "\n\r" بمعنى أنه لا يتم الانتقال إلى أول السطر إذا استقبلت اللبنة "\n" .

وسوف نلتقى بهذه الدوال واستخداماتها لقاء آخر عند الحديث عن النوافذ في نسق الكتابة .

(٢ - ٩) دالة مسح الشاشة أو النافذة `clrscr`

تستخدم الدالة `clrscr` لمسح النافذة (أو الشاشة) في بيئة الكتابة ويعقب ذلك ظهور مؤشر الكتابة (`cursor`) في الركن العلوى الأيسر من الشاشة أى عند الإحداثى (1,1) . والدالة `clrscr` معرفة في ملف العناوين "`conio.h`" . وتأخذ عينة الدالة الصورة الآتية :

```
#include <conio.h>

void clrscr(void);
```

شكل (٢٧)

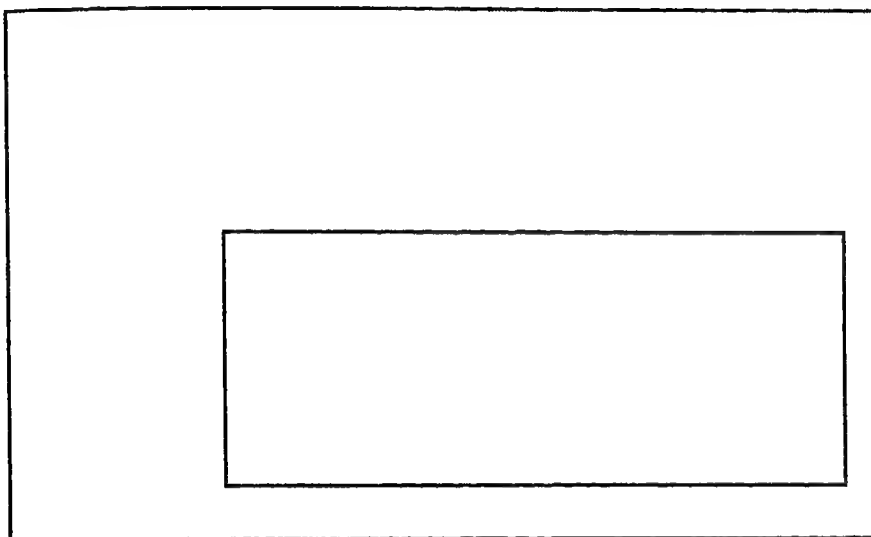
(٢ - ١٠) النوافذ في نسق الكتابة `window`

إن شاشة الكتابة عبارة عن نافذة محددة بالإحداثيات :

(1,1,80,25)

بمعنى أن الركن الأيسر العلوى هو الإحداثى (1,1) والركن الأيمن السفلى هو الإحداثى (80,25) ، هذا بالنسبة للأطوار `B80` ، `C80` . هذه هى النافذة سابقة التعريف التى نستخدمها فى الكتابة والتى لا يجوز الخروج عن حدودها . ويمكنك باستخدام الدالة `window` أن تنشئ نافذة أصغر ذات أركان محددة بداخل النافذة الكبيرة (الشاشة) وفى هذه الحالة يمكنك إجراء عمليات

إدخال وطباعة البيانات بداخل حدود هذه النافذة باستخدام الدوال الموجهة إلى النوافذ .



شكل (٢٨)

والدالة window تأخذ الصورة الآتية :

```
#include <conio.h>

void window(int left, int top,
            int right, int bottom);
```

شكل (٢٩)

حيث :

- left هو الإحداثي الأفقي للركن الأيسر العلوي (العمود) .
- top هو الإحداثي الرأسى للركن الأيسر العلوي (الصف) .
- right هو الإحداثي الأفقي للركن الأيمن السفلى (العمود) .

bottom هو الإحداثى الرأسى للركن الأيمن السفلى
(الصف) .

وبمجرد استدعاء هذه الدالة لإنشاء نافذة معينة فإن جميع الإحداثيات تصبح
منسوبة إلى النافذة بدلاً من الشاشة .

فعلى سبيل المثال تؤدي استخدام الدالة gotoxy إلى تحريك مؤشر الكتابة
بداخل النافذة كما يؤدي استخدام الدالة clrscr إلى مسح النافذة فقط بصرف
النظر عن محتويات الشاشة .

ولنجرب معاً تنفيذ البرنامج التالى الذى يوضح لنا مفهوم النوافذ . وعند
تشغيل البرنامج تتابع الأحداث الآتية :

١ — يبدأ البرنامج بطباعة عبارة تحية على النافذة سابقة التعريف (الشاشة)
باستخدام دالة الطباعة القياسية printf .

٢ — عند الضغط على أى زر ، تنشأ النافذة الأولى وبها رسالة تحية من
داخل النافذة .

٣ — عند الضغط على أى زر ، تمسح محتويات النافذة الأولى (بينما تظل
محتويات الشاشة كما هى) وتظهر النافذة الثانية وبها رسالة جديدة نخبرنا عن
تمام إنشاء النافذة الثانية .

٤ — عند الضغط على أى زر تمسح محتويات النافذة الثانية وننتقل إلى
الشاشة مرة أخرى مع رسالة تحية جديدة . وبذلك ففى نهاية البرنامج نرى
فقط الرسالة الأولى والرسالة الأخيرة أما محتويات النوافذ فتختفى . انظر
الأشكال التالية :


```

/* Program 2-7:cpp */
#include <conio.h>
#include <stdio.h>
/* Macros */
#define WINDOW1() window(20,5,60,10); \
                        textcolor(YELLOW);
#define WINDOW2() window(20,11,60,20); \
                        textcolor(MAGENTA);
main()
{
    char *a, *b, *c, *d, *e;
    a="Hi There, I am in window #1.";
    b="Press any key..";
    c="Hello again, I am in window #2.";
    d="Hello, I am in the default window, the screen."
    e="Hello there... \nNow I am outside windows..";
    /* Clear the screen */
    clrscr();
    printf("\n%s\n%s",d,b);
    getch();
    /* Create window #1 */
    WINDOW1();
    cprintf("%s\n\r%s",a,b);
    getch();
    /* Clear window #1 */
    clrscr();
    /* Create window #2 */
    WINDOW2();
    cprintf("%s\n\r%s",c,b);
    getch();
    /* Clear window #2 */
    clrscr();
    printf("\n\n%s\n%s",e,b);
    getch();
    return(0);
}

```

شكل (٣٢)
البرنامج السابع

مناقشة البرنامج : (ملاحظة : استعن بالأرقام فى تتبع البرنامج)

[١] يبدأ البرنامج بإعلان دوال النوافذ كما كرو :

WINDOW1()	لنافذة الأولى
WINDOW2()	لنافذة الثانية

ونلاحظ أن الماكرو قد شمل استدعاء الدالة window علاوة على تحديد اللون المستخدم للحروف بالنافذة وذلك باستخدام الدالة textcolor .

[٢] نلاحظ أن الدالة clrscr التى جاءت فى بداية البرنامج تؤدي إلى مسح الشاشة كلها (قبل دخول النوافذ) . أما بعد إنشاء النافذة الأولى فإن نفس الدالة تؤدي إلى مسح النافذة الحالية فقط دون المساس بمحتويات الشاشة . وكذلك الحال بالنسبة للنافذة الثانية .

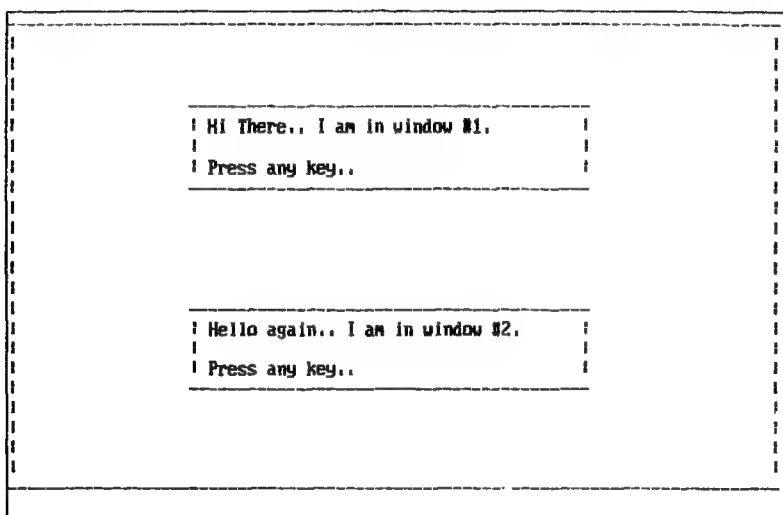
[٣] نلاحظ أيضاً أن بعض عبارات الطباعة جاءت محتوية على الدالة printf والأخرى جاءت محتوية على الدالة cprintf . وهذا مثال جيد لتوضيح الفارق بين الدالتين .

فالدالة الأولى تتعامل مع الشاشة دائماً حتى لو كانت هناك نافذة مفتوحة فهي لا تنتمى إلى دوال نسق الكتابة . أما الدالة الثانية فهي تتعامل دائماً مع النافذة المفتوحة ، ولذلك نرى أن العبارات المطبوعة بداخل النوافذ جاءت بادئة من حدود النافذة وليس من أقصى اليسار . ولا يغوتنا ملاحظة أننا فى هذا البرنامج لم نستخدم الدالة gotoxy بل اكتفينا بالحركة سابقة التعريف لمؤشر الكتابة بداخل النوافذ .

أما البرنامج التالى فنعمق فيه مفهوم النوافذ بتحديد إطار النافذة وذلك بإنشاء دالة جديدة فى البرنامج وهى الدالة "frame" التى تستخدم أربعة بارامترات ممثلة لأركان النافذة . ويتم رسم الإطار باللبنات آسكى المعتادة (باستخدام الشرط الأفقية والشرط الرأسية) .

كما تم استخدام الدالة gotoxy في هذا البرنامج لطباعة الرسائل في المكان المناسب بالنافذة .

والشكل التالي يوضح التنفيذ المتوقع للبرنامج :



شكل (٣٣)



```

/* Program 2-8.cpp */
#include <conio.h>
#include <stdio.h>
// Outside frame coordinates:
#define m1 1
#define n1 1
#define m2 80
#define n2 25
// First window coordinates:
#define a1 20
#define b1 5
#define a2 60
#define b2 10
// Window Macros
#define WINDOW1() window(a1,b1,a2,b2);
#define WINDOW2() window(a1,b1+10,a2,b2+10);
// Prototypes
void frame(int, int, int ,int);
main()
{
    char *a, *b, *c, *d;
    a="Hi There.. I am in window #1.";
    b="Press any key..";
    c="Hello again.. I am in window #2.";
    clrscr();
    // Draw the frame of the screen:
    frame(m1,n1,m2,n2);
    // Create window #1:
    WINDOW1();
    clrscr();
    // The frame color for window #1:
    textcolor(WHITE);
    // The frame of window #1:
    frame(a1, b1, a2, b2);
    // The text color for window #1:
    textcolor(RED);
    gotoxy(3,2);                // the window starts at 1,1
    cprintf("%s",a);
    gotoxy(3,4);
    cprintf("%s",b);
    getch();
    // Create window #2:
    WINDOW2();
    clrscr();

```

شكل (٣٤)
الجزء الأول من البرنامج الثامن

```

// The frame color for window #2:
    textcolor(YELLOW);
// The frame of window #2:
    frame(a1, b1+10, a2, b2+10);
// The text color for window #2:
    textcolor(CYAN);
    gotoxy(3,2);
    cprintf("%s",c);
    gotoxy(3,4);
    cprintf("%s",b);
    getch();
    return(0);
}
// Frame function:
void frame(int x1, int y1, int x2, int y2)
{
    register int i;
    gotoxy(1,1);          // Horizontal line - top
    for (i=0; i<= x2-x1; i++)
        putchar('-');
    gotoxy(1, y2-y1); // Horizontal line - bottom
    for (i=0; i<= x2-x1; i++)
        putchar('-');
    // Vertical lines:
    for (i=2; i< y2-y1; i++) {
        gotoxy(1, i);          // left
        putchar('|');
        gotoxy(x2-x1+1, i);    // right
        putchar('|');
    }
}

```

شكل (٣٥)
الجزء الثاني من البرنامج الثامن

مناقشة البرنامج : (ملاحظة : استعن بالأرقام لتتبع البرنامج)

[١] يبدأ البرنامج بإعلان مجموعة من الثوابت الممثلة لإحداثيات إطار الشاشة (1,1,80,25) وكذلك إحداثيات إطار النافذة الأولى (20,25,60,10) . وباستخدام ثوابت النافذة الأولى مع إزاحة مناسبة يمكن بناء إطار النافذة الثانية بنفس الأبعاد وفي موقع جديد .

[٢] يلي ذلك تعريف ماكروا التوافذ ، وقد أتى التعريف هذه المرة بدون الألوان ، حيث أننا سنخصص لوناً للإطار ولوناً للكتابة بكل نافذة .

[٣] تلا ذلك إعلان عينة الدالة "frame" المخصصة لرسم الإطارات .

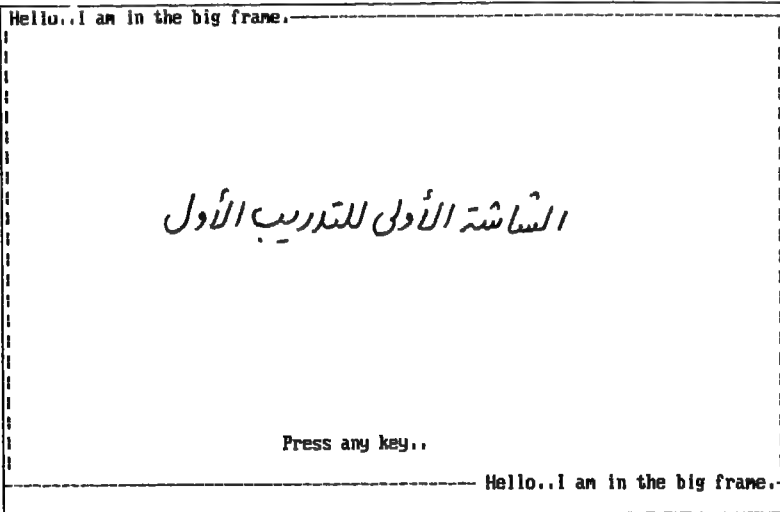
[٤] ثم بعد ذلك إنشاء التوافذ والكتابة بداخلها بالطريقة المعتادة مع استخدام الدالة gotoxy لتوجيه مؤشر الكتابة إلى المكان المناسب بالنافذة .

[٥] نلاحظ في دالة الإطار (frame) أننا استخدمنا دالة طباعة اللينات المخصصة للتوافذ putch لرسم البرواز ، وهى الدالة الوحيدة التى تصلح لهذا الغرض .

ولو أننا جربنا استخدام دالة بديلة مثل الدالة القياسية putchar فسوف نجد أنها لا تستجيب للألوان ، وتستخدم دائماً اللون سابق التعريف للكتابة .

تدريب (٢ - ١)

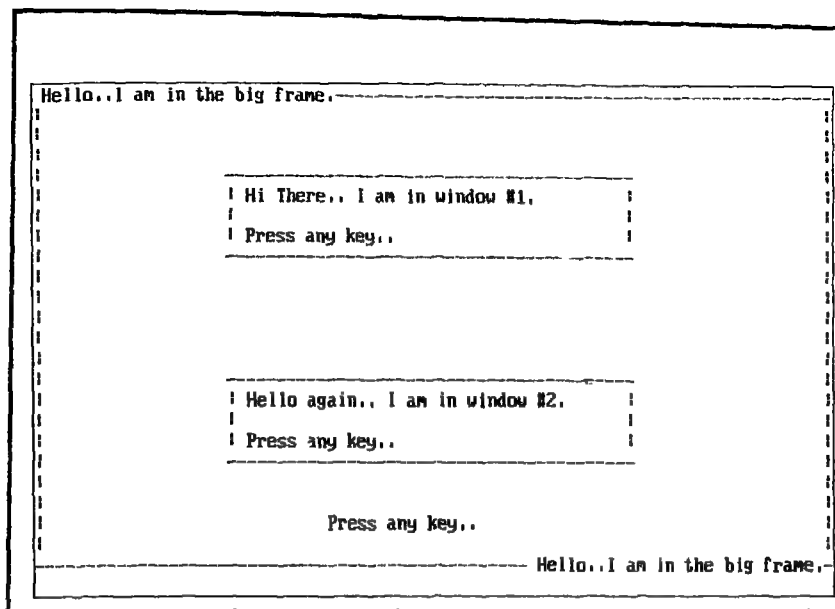
أجر التعديلات اللازمة على البرنامج السادس (2-6.cpp) بحيث تظهر مبدئياً الشاشة الموضحة بعد وعليها العبارات الموضحة في الأماكن المحددة .



شكل (٣٦)

وعند الضغط على أى زر يتم الانتقال إلى الشاشة التالية حيث تظهر النافذة الأولى بمحتوياتها ، فإذا ضغطت على زر جديد ظهرت النافذة الثانية بحيث تظهر النافذتان والإطار الكبير للشاشة في نفس الوقت .

وعند الضغط على أى زر يختفى المشهد وينتهى البرنامج . تظهر جميع الألوان في هذا البرنامج عالية الإضاءة .



شكل (٣٧)

(٢ - ١١) استخدام النوافذ في التطبيقات

إن استخدام النوافذ في بيئة الكتابة يعد طريقة سهلة لإعداد الوصلة البينية للمستخدم في البرامج التطبيقية .

ورغم أن النوافذ التي أنشأناها حتى الآن لا تتمتع بالمظهر الجيد لكن إضافة لمسة الجمال لا يحتاج إلى أى جهد أبعد من ذلك . فلو أنك في أى نافذة ، قد حددت لون الخلفية ثم استخدمت دالة مسح النافذة `clrscr` فسوف تتحول النافذة إلى مساحة ملونة ، وفي هذه الحالة لا نحتاج إلى الإطار على الإطلاق لإظهار حدود النافذة .

وفي البرنامج التالى سوف نقوم بتصميم القائمة الموضحة بالشكل التالى ، والتي يمكن استخدامها في إدخال بيانات قاعدة بيانات لأرقام التليفونات ، كما يجوز تطوير البرنامج لاستخدامه في أى غرض آخر .

- 1- Enter name.
- 2- Enter phone number.
- 3- Exit.

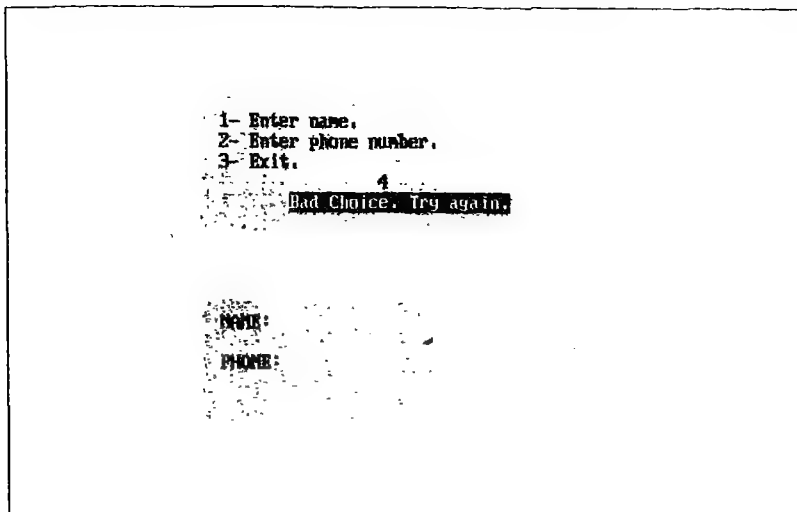
NAME: Sam A. Abolrous

PHONE: 651-6052

شكل (٣٨)

والشاشة التي نراها بالشكل تحتوي على نافذتين ؛ واحدة لعرض القائمة والاختيارات المتاحة بها ، والثانية لاستقبال البيانات من مستخدم البرنامج . وعندما تختار من النافذة الأولى أحد الاختيارات بالضغط على زر الرقم المناسب فإن هذا الرقم يظهر بلون مميز في وسط النافذة ثم يقفز مؤشر الكتابة إلى الموضع المناسب في النافذة الثانية . فاختيار الرقم 1 مثلاً يجعل مؤشر الكتابة يقفز إلى سطر الاسم (NAME) جاهزاً على استقبال بيان الاسم . وإذا ان الرقم المختار هو 2 قفز المؤشر إلى السطر الثاني (PHONE) جاهزاً على استقبال رقم التليفون . أما الاختيار 3 فهو ينهي البرنامج .

ماذا يكون الحال لو أنك ضغطت على رقم بخلاف الأرقام الواردة في القائمة (3,2,1) ؟ في هذه الحالة تظهر رسالة خافقة (blinking) تنبه المستخدم إلى الخطأ الحادث وتطلب منه إعادة المحاولة ، كما في الشكل التالي ، فإذا كانت المحاولة التالية سليمة اختفت الرسالة وعاد البرنامج إلى مجراه الطبيعي .



شكل (٣٩)

أما عن ألوان النوافذ ، فالنافذة الأولى ذات خلفية بيضاء وتظهر فيها العناصر باللون الأحمر . أما الاختيار الذي يُدخله المستخدم (مثل الرقم 1 أو 2) فيظهر باللون الأرجواني (MAGENTA) . أما الرسالة المخافة فحروفها صفراء على خلفية حمراء .

أما النافذة الثانية فهي ذات خلفية تيركوازية (CYAN) تظهر بها العناوين الثابتة مكتوبة باللون الأحمر . أما البيانات التي يدخلها المستخدم فهي تظهر باللون الأسود . فلنجرّب البرنامج ثم نلتقى حول المناقشة .



```

/* Program 2-9.cpp */
#include <conio.h>
#include <stdlib.h>
// First window coordinates:
#define a1 20
#define b1 5
#define a2 60
#define b2 11
// Windows definitions:
#define WINDOW1() window(a1,b1,a2,b2); \
    textcolor(MAGENTA); textbackground(WHITE);
#define WINDOW2() window(a1,b1+10,a2,b2+10); \
    textcolor(MAGENTA); textbackground(CYAN);
main()
{
    char c;
    char name[25+2], phone[9+2];
    // Initialize element 0 to max length:
    name[0]=25 ;phone[0]=9;
    // Define mode and background:
    textmode(C80);
    textbackground(BLACK);
    // Clear the screen:
    clrscr();
    // Create first window:
    WINDOW1();
    // Clear the window:
    clrscr();
    // Display window info. in higvideo:
    textcolor(RED);
    gotoxy(3,2);
    cputs("1- Enter name.");
    gotoxy(3,3);
    cputs("2- Enter phone number.");
    gotoxy(3,4);
    cputs("3- Exit.");
    // Create second window:
    WINDOW2();
    // Clear the window:
    clrscr();
    // Display window info. in higvideo:
    textcolor(RED);
    gotoxy(3,2);
    cputs("NAME:");

```

شكل (٤٠)

الجزء الأول من البرنامج التاسع

```

        gotoxy(3,4);
        cputs("PHONE:");
// Back to window #1:
        WINDOW1();
        gotoxy(19,5);
// Receive input inside window #1:
        c=getche();
        for(;;) {
            switch(c)
// Accept data in window #2:
            { case '1':
                WINDOW2();
                textcolor(BLACK);
                gotoxy (10,2);
                cgets(name);
                break;
              case '2':
                WINDOW2();
                textcolor(BLACK);
                gotoxy (10,4);
                cgets(phone);
                break;
              case '3':
                textmode(C80);
                clrscr();
                exit(0);
              default:
// Display a blinking error message:
                textattr(128 | YELLOW | RED*16);
                gotoxy(10,6);
                cputs("Bad Choice. Try again.");
            }
// Get a new choice:
            WINDOW1();
            gotoxy(19,5);
            clreol();
            c=getche();
// Clear the error message, if any:
            gotoxy(10,6);
            clreol(); ←
        }
    }
}

```

شكل (٤١)

الجزء الثاني والأخير من البرنامج التاسع

مناقشة البرنامج

[١] يبدأ البرنامج كالمعتاد بتحديد أبعاد النافذة الأولى a_1, b_1, a_2, b_2

[٢] يلى ذلك إعلانات ماكرو النوافذ وهى تشمل لون الحروف ولون الخلفية . أما لون الحروف الوارد فى الماكرو فهو يستخدم كلون سابق التعريف ولكنه يجوز تغييره بداخل البرنامج نفسه كما سنرى .

فالنافذة الأولى مثلاً تستخدم اللون الأرجوانى للحروف واللون الأبيض للخلفية .

```
// Windows definitions:
#define WINDOW1() window(a1,b1,a2,b2); \
    textcolor(MAGENTA); textbackground(WHITE);
#define WINDOW2() window(a1,b1+10,a2,b2+10); \
    textcolor(MAGENTA); textbackground(CYAN);
```

شكل (٤٢)

ولذلك فعند كل استدعاء لهذا الماكرو سوف تستدعى الألوان مع النافذة حتى لو كان اللون قد تم تغييره أثناء البرنامج .

[٣] تأتى بعد ذلك العبارات المختلفة لكتابة النصوص المختلفة فى النوافذ وليس بها جديد سوى استخدام الدالة clrscr بعد الاستدعاء الأول لكل نافذة والتي بمقتضاها يتم تلوين خلفية النافذة وهذا يعطينا عن استخدام إطار النافذة (غير مأسوف عليه) .

[٤] تم عملية استقبال اختيار القائمة من المستخدم باستخدام الدالة getch وتتم عملية اختبار الرقم المختار باستخدام الدالة switch .

وينقسم بلوك الدالة switch إلى أربع حالات :

© إذا كان الاختيار هو الرقم 1 :

فى هذه الحالة تستدعى النافذة الثانية ، ويقفز المؤشر إلى الموضع

(10,2) أمام كلمة "NAME" مع تلوين الكتابة باللون الأسود تمهيداً لاستقبال بيان الاسم .

```
PP: switch(c)
// Accept data in window #2:
{ case '1':
    WINDOW2();
    textcolor(BLACK);
    gotoxy (10,2);
    cgets(name);
    break;
```

شكل (٤٣)

© إذا كان الاختيار هو الرقم 2 :

في هذه الحالة يقفز مؤشر الكتابة إلى النافذة الثانية إلى الموضع (10,4) أى أمام كلمة "PHONE" تمهيداً لاستقبال رقم التليفون مع تلوين الحروف باللون الأسود .

```
case '2':
    WINDOW2();
    textcolor(BLACK);
    gotoxy (10,4);
    cgets(phone);
    break;
```

شكل (٤٤)

© إذا كان الاختيار هو الرقم 3 :

يتم في هذه الحالة الخروج من البرنامج بالدالة exit(0) مع تنظيف الشاشة من محتوياتها بما في ذلك النوافذ .

```
case '3':
    textmode(C80);
    clrscr();
    exit(0);
```

شكل (٤٥)

© الاختيارات الأخرى :

إذا كان الرقم المختار خلاف الأعداد الثلاثة 1,2,3 فإن رسالة الخطأ الآتية تظهر في النافذة عند الإحداثي (10,6) :

Bad Choice. Try again

```
default:
// Display a blinking error message: الحارات النظرية
textattr(128 | YELLOW | RED*16);
gotoxy(10,6);
cputs("Bad Choice. Try again.");
```

شكل (٤٦)

ويقع المنشأ "سويتش" بالكامل بداخل حلقة تكرارية لانتهائية .

[٥] في جميع الأحوال — ما لم يتم الخروج من البرنامج — فإنه عند الخروج من المنشأ "سويتش" بالدالة break فإنه يتم مسح الاختيار السابق (رقم الاختيار 1 أو 2 أو 3) تمهيداً لاستقبال اختيار جديد ، وذلك باستخدام الدالة :

clreol

ويأتى اسم هذه الدالة من العبارة :

clear up to end of line

وهي تؤدي إلى مسح محتويات النافذة الواقعة بين الموقع الحالي لمؤشر الكتابة وحتى آخر السطر (بداخل النافذة) .

وعينة هذه الدالة تأخذ الصيغة الآتية :

```
#include <conio.h>
```

```
void clreol(void);
```

شكل (٤٧)

وتستخدم الدالة clreol أيضاً عند مسح رسالة الخطأ .

```
// Get a new choice:
WINDOW1();
gotoxy(19,5);
clreol();
c=getche();
// Clear the error message, if any:
gotoxy(10,6);
clreol();
```

شكل (٤٨)

[٦] لم نناقش بعد طريقة إدخال الحرفيات إلى البرنامج وقد أرجأنا ذلك للنهاية حيث ظهرت دالة جديدة هنا لاستقبال الحرفيات ، وهي الدالة :

cgets

والدالة cgets ذات نفع خاص في مجال النوافذ فهي تستقبل اللبئات المتتابة للحرفي ثم تتوقف عندما يصل عدد اللبئات إلى الحد الأقصى السابق تحديده في البرنامج . وفي حالتنا هذه فقد حددنا طول الاسم بعدد ٢٥ حرفاً ، يُستخدم منها ٢٤ حرفاً وتترك اللبئة الأخيرة للزر Enter .

وتأخذ عينة الدالة cgets الصورة الآتية (بالملف conio.h) :

```
#include <conio.h>

char *cgets(char *str);
```

شكل (٤٩)

وعندما يتم قراءة الحرفي بهذه الدالة فإنها تقوم بتخزينه وتخزين طوله في خانة الذاكرة المشار إليها بالمؤشر Str . ومن اللازم عند استخدام هذه الدالة أن يتم تخزين أقصى طول للحرفي في العنصر الأول من مصفوفة اللبئات Str[0] وهذا نراه في مستهل البرنامج .


```
char c;
char name[25+2], phone[9+2];
// Initialize element 0 to max length:
name[0]=25; phone[0]=9;
```

شكل (٥٠)

وعندما تضغط على الزر ENTER فإن المجموعة " \n\r " تتحول إلى اللبنة " \0 " التي تضاف إلى مؤخرة الحرفي قبل تخزينه .

وعند رجوع الدالة يصبح العنصر الثاني Str[1] محتوياً على طول الحرفي الحقيقي الذي تمت قراءته بالفعل . أما الحرفي نفسه فيخترن في بقية المصفوفة بدءاً من العنصر Str[2] . وهذا هو السبب في أن سعة المصفوفة name (وكذلك phone) جاءت أبعادها [25+2] أى تزيد بمقدار 2 عن أقصى طول للحرفي المتوقع .

أما القيمة المرتجعة من هذه الدالة فهي مؤشر إلى العنصر الثالث str[2] الذي يبدأ عنده الحرفي .

وإذا أردت طباعة محتويات الحرفي فيلزم تخصيص الدالة cgets إلى مؤشر لبنة ثم طباعة محتويات المؤشر كالتالي :

```
char *p;
...
p=cgets(name);
...
printf("%s",p);
```

شكل (٥١)

الموجز

[١] فى هذا الباب انتقلنا من نسق الرسم إلى نسق الكتابة وقد بدأنا باستخدام الدالة :

closegraph

لإنهاء نسق الرسم وتحرير الذاكرة المشغولة بأدواته .

[٢] وفى نسق الكتابة قد تعرفنا بمهارات مختلفة تشمل تحريك مؤشر الكتابة إلى مواقع محددة على الشاشة ، وتلوين الحروف وخلفيتها .

[٣] كما عرفنا كيفية إنشاء النوافذ فى نسق الرسم واستخدامها فى تصميم القوائم المستخدمة فى البرامج التطبيقية . وفى هذا السياق عرفنا الكثير من دوال نسق الكتابة الموجهة إلى النوافذ .

[٤] فيما يلى ملخص بالدوال التى التقينا بها فى هذا الباب :

#include <graphics.h>	دالة إنهاء نسق الرسم
void far closegraph(void);	
#include <conio.h>	تحريك مؤشر الكتابة
void gotoxy(int x, int y);	
#include <conio.h>	تغيير طور الكتابة
void textmode(int mode);	
#include <conio.h>	الكتابة فى النافذة
int cprintf(const char *format [,argument,...]);	
#include <conio.h>	الكتابة فى النافذة
int cputs(const char *str);	
#include <conio.h>	تغيير لون الكتابة
void textcolor(int newcolor);	
#include <conio.h>	تغيير لون خلفية الكتابة
void textbackground(int newcolor);	

```

#include <conio.h>                تغيير صفت الكتابة
void textattr(int newattr);

#include <conio.h>
void highvideo(void);             تغبي مستوى الإضاءة
void lowvideo(void);
void normvideo(void);

#include <conio.h>                 إدخال لبنة من لوحة الأزرار
int getche(void);

#include <conio.h>                 كتابة لبنة بالنافذة
int putch(int ch);

#include <conio.h>                 مسح النافذة
void clrscr(void);

#include <conio.h>                 إنشاء النافذة
void window(int left, int top,
             int right, int bottom);

#include <conio.h>                 إدخال حرفي من لوحة الأزرار
char *cgets(char *str);

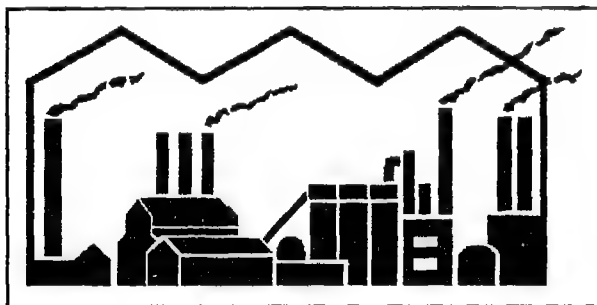
#include <conio.h>                 المسح حتى نهاية السطر
void clreol(void);                بالنافذة الحالية .

```



الباب الثالث

نقط وخطوط ودوائر



مفتّح

فى هذا الباب نستعرض الطرق المختلفة لرسم الأشكال الهندسية . وهذه الأشكال مهما بلغت درجة تعقيدها فهى تعتمد على أحجار ثلاثة من أحجار البناء : النقطة والخط والدائرة . ويأتى ضمن الأشكال الهندسية تمثيل بعض الدوال الرياضية مثل دالة الموجة الجيبية (sine) .

كما يندرج تحت نفس العنوان الأسطح والأشكال المجسمة .

كما سنعرض فى هذا الباب كيفية التحكم فى شكل وسمك الخطوط التى تنتجها دوال الرسم المختلفة .

(٣ - ١) الأشكال الهندسية

تعرفنا في الباب الأول ببعض دوال رسم الأشكال الهندسية مثل الدائرة والمستقيم . ومكتبة دوال الرسم بلغة سي/سي++ غنية بدوال الأشكال الهندسية التي لا يستغنى عنها مبرمج مثل القوس والشكل المضلع والقطع إلى آخره ، كما يمكنك باستخدام هذه المجموعة الأساسية بناء الأشكال المعقدة مثل السطوح والأشكال المجسمة .

وقبل الدخول في الموضوعات الجديدة سوف نقدم بعض المهارات العامة التي يمكن الاستفادة بها في نسق الرسم . ولنبدأ ببعض الأمثلة على استخدام الدوائر والخطوط .

التعرف الآتوماتيكي على أبعاد الشاشة
getmaxx
getmaxy

في البرنامج التالي نرسم مجموعة من الدوائر متحدة المركز وذلك بتغيير نصف قطر الدائرة خلال حلقة تكرارية . وقد استخدمنا نقطة مركز الشاشة كمركز لمجموعة الدوائر . والجديد في هذا البرنامج هو الدالتان المستخدمتان في إيجاد مركز الشاشة :

getmaxx
getmaxy

فمن المعروف أن مركز الشاشة (بالنسبة للطور VGAHI) يقع عند الإحداثي (639/2, 479/2) . ولكننا لو استخدمنا طوراً مختلفاً للرسم فسوف تتغير دقة الشاشة وتتغير إحداثيات المركز ، وبالتالي فإن البرنامج الذي يستخدم الأرقام الصريحة سوف يؤدي إلى نتائج مختلفة مع أطوار الرسم ذات الدقة المختلفة .

أما استخدام الدوال `getmaxx` ، `getmaxy` فهو يغنينا عن هذه المتاعب كما يغنينا من الحسابات أصلاً ؛ فهذه الدوال تقوم باستكشاف أبعاد الشاشة وفقاً للدقة المستخدمة وترجع أقصى قيمة للإحداثي x وأقصى قيمة للإحداثي y .

معنى ذلك أن إحداثيات مركز الشاشة هي دائماً :

- الإحداثي الأفقي (نصف الاتساع) `getmaxx()/2`
- الإحداثي الرأسى (نصف الارتفاع) `getmaxy()/2`

وعينات هذه الدوال موجودة بملف عناوين مكتبة الرسم "`graphics.h`" وتأخذ الصورة :

```
#include <graphics.h>
int far getmaxx(void);
int far getmaxy(void);
```

شكل (١)

وقد تم تخصيص الإحداثي الأفقي الممثل لنصف اتساع الشاشة للمتغير "`HalfWidth`" . وتم تخصيص الإحداثي الرأسى الممثل لنصف ارتفاع الشاشة للمتغير `HalfHeight` .

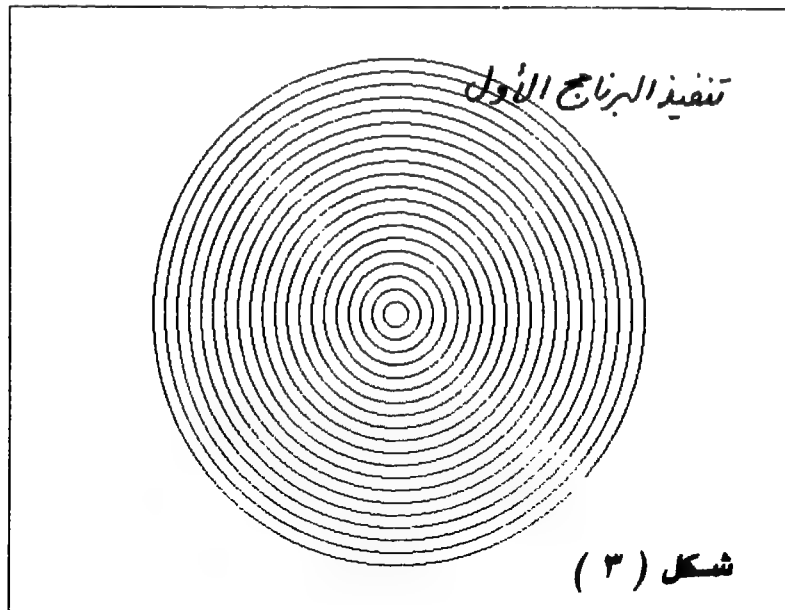
وقد أضفنا إلى البرنامج حلقة تكرارية للتأخير حتى يمكنك مشاهدة الدوائر وهي ترسم واحدة بعد الأخرى . وإذا كنت تستخدم كومبيوتر ذا سرعة محدودة فمن الأفضل أن تقلل فترة التأخير المستخدمة أو تحذفها على الإطلاق . وهذا هو البرنامج :


```

/* Program 3-1.cpp */
// Concetric circles
#include <graphics.h>
#include <conio.h>
main()
{
    int driver, mode;
    int HalfHeight, HalfWidth;
    int x, y, r;
    long loop;
    driver = DETECT;
    initgraph(&driver, &mode, "D:/TC/BGI");
    setcolor(CYAN);
    HalfHeight = getmaxy()/2;
    HalfWidth = getmaxx()/2;
    for (r=10; r<=200; r=r+10) {
        circle(HalfWidth,HalfHeight,r);
        for (loop=1; loop<=100000; loop++);
    }
    getch();
    return (0);
}

```

شكل (٢) البرنامج الأول



فلاش

عند استخدام دوال تحديد أبعاد الشاشة :

`getmaxx`

`getmaxy`

فإنه من اللازم أن تُستدعى هذه الدوال - كسائر دوال الرسم - بعد دخول نسق الرسم بالدالة `initgraph` وإلا فإنها تعطى القيمة صفراً .

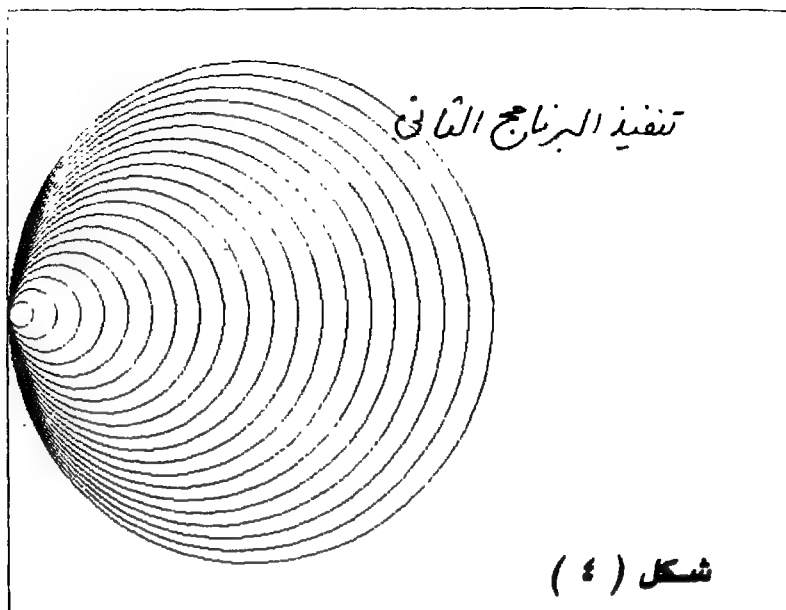
(٣ - ٢) الأشكال الدائرية

تحريك دائرة على خط مستقيم

يمكنك باستخدام البرنامج الأول وبإجراء بعض التعديلات الحصول على أشكال كثيرة باستخدام الدوائر .

فلاسطوانة مثلاً ، ما هي إلا دائرة متحركة على خط مستقيم مع الاحتفاظ بنفس نصف القطر . ولو أنك غيرت نصف القطر بانتظام أثناء حركة الدائرة فسوف تحصل على مخروط مفتوح . ولو بدأت بدائرة قطرها صفراً تحصل على مخروط مقفل .

وفي البرنامج التالي سوف نحرك مركز الدائرة على خط أفقى بدءاً من أقصى اليسار مع زيادة نصف قطرها بخطوة معلومة فنحصل على الشكل الموضح بعد للدوائر غير المتمركزة .



```

/* Program 3-2.cpp */
// Eccentric circles
#include <graphics.h>
#include <conio.h>
main()
{
    int driver, mode;
    int HalfHeight;
    int x, y, r;
    long loop;
    driver = DETECT;
    initgraph(&driver, &mode, "D:/TC/BGI");
    setcolor(YELLOW);
    HalfHeight = getmaxy()/2;
    // The graph:
    for (r=10; r<=200; r=r+10) {
        circle(r, HalfHeight, r);
        for (loop=1; loop<=100000; loop++);
    }
    getch();
    return (0);
}

```

شكل (٥)
البرنامج الثاني

وكما نرى فى البرنامج فإن الدائرة تبدأ بنصف قطر :

$$r=10$$

ثم يتم زيادته بمقدار 10 فى كل دورة من دورات الحلقة التكرارية حتى يصل إلى القيمة 200 .

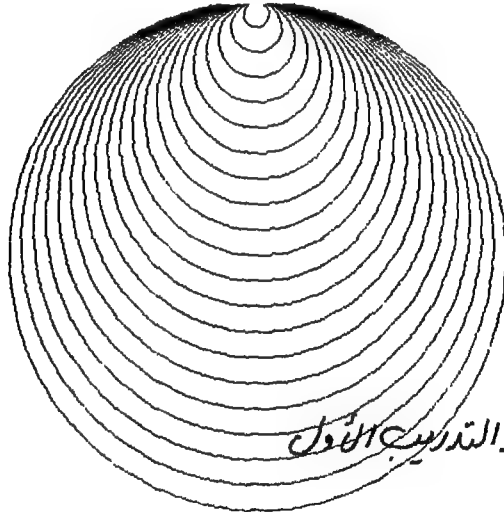
كما نلاحظ أن الدائرة ترسم عند الإحداثيات :

(r, HalfHeight)

معنى ذلك أن الإحداثى الأفقى يتغير بنفس القيمة كما نصف القطر أما الإحداثى الرأسى فهو ثابت دائماً ويأخذ قيمة نصف ارتفاع الشاشة .

تدريب (١ - ٣)

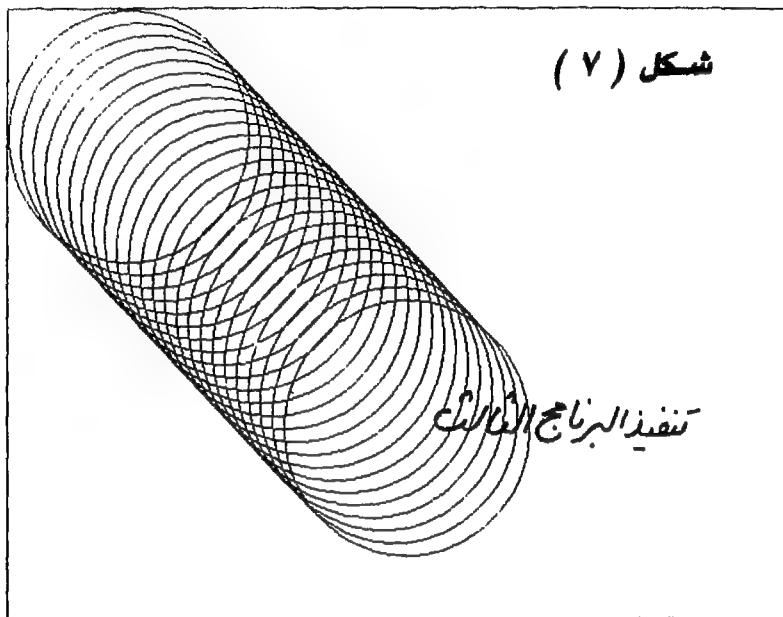
أجر التعديلات اللازمة على البرنامج الثانى لكى تحصل على الشكل التالى (دائرة يتزايد نصف قطرها ويتحرك مركزها رأسياً) :



تنفيذ التدريب الأول

شكل (٦)

أما البرنامج التالي فهو يرسم اسطوانة وذلك بتحريك الدائرة في اتجاه قطر الشاشة وذلك بزيادة كل من الإحداثي الأفقي والإحداثي الرأسى بنفس النسبة :

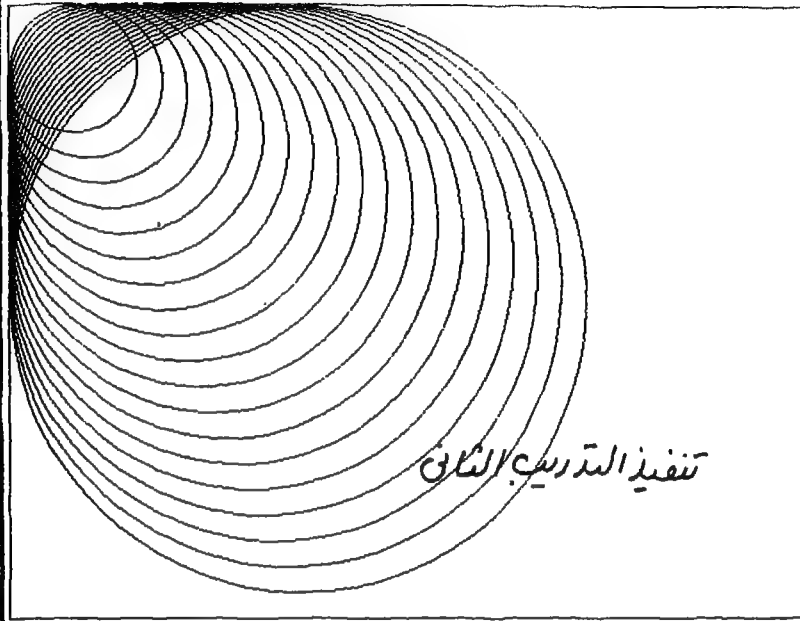


```
/* Program 3-3.cpp */
// Cylinder
#include <graphics.h>
#include <conio.h>
main()
{
    int driver, mode, r;
    long loop;
    driver = DETECT;
    initgraph(&driver, &mode, "D:/TC/BGI");
    setcolor(YELLOW);
    for (r=100; r<=350; r=r+10) {
        circle(r,r,100);
        for (loop=1; loop<=100000; loop++);
    }
    getch();
    return(0);
}
```

شكل (٨)

تدريب (٣-٢)

اكتب برنامجاً لرسم الشكل الموضح بعد :



شكل (٩)

تحريك دائرة على محيط دائرة أخرى

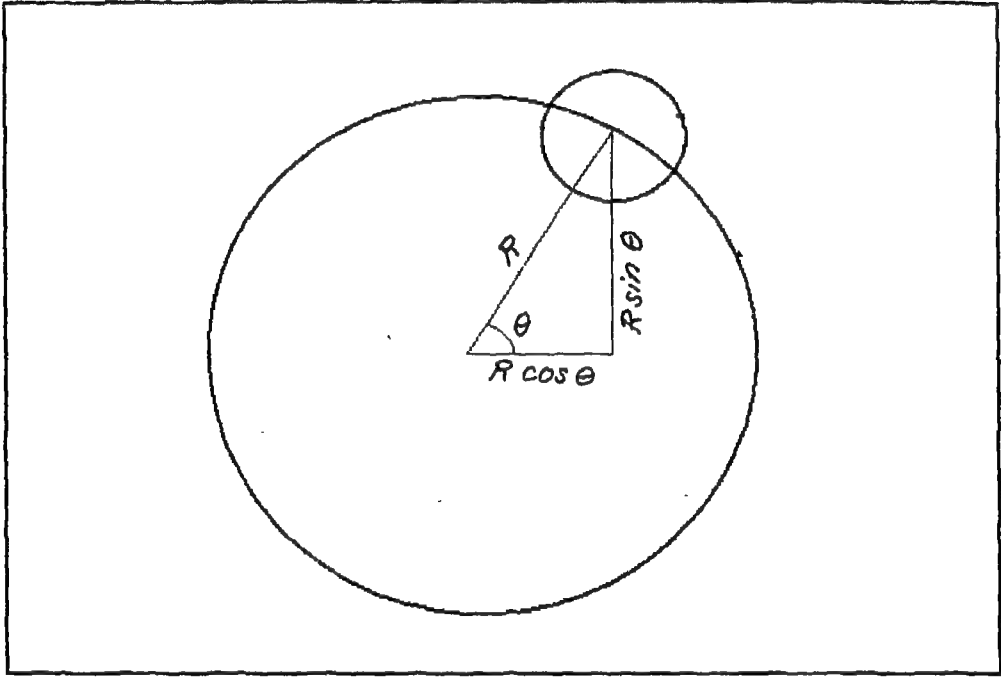
إن شكل الكعكة وإطار السيارة وطوق النجاة عبارة عن دائرة متحركة على محيط دائرة أخرى . ويطلق على هذا الشكل في البرنامج الهندسي AutoCAD الاسم "Downut" .

ولكى نحرك الدائرة على محيط دائرة أخرى يلزمنا أن تكون إحداثيات مركز الدائرة المتحركة كالتالي :

$$x = R \cdot \cos \theta$$

$$y = R \cdot \sin \theta$$

حيث R هو نصف قطر الدائرة الأخرى أما الزاوية θ فهي متغير يمكننا التحكم فيه لتحريك الدائرة عدداً معيناً من الدرجات (كعكسة كاملة أو جزء من عكسة) . انظر الشكل التالى الذى يوضح علاقة مركز الدائرة المتحركة بالدائرة الثابتة .

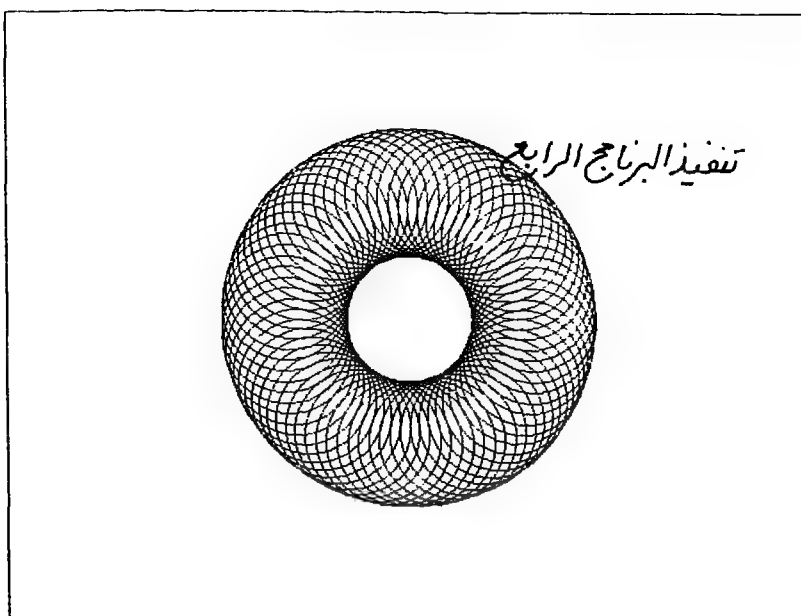


شكل (١٠)

حركة دائرة على محيط دائرة أخرى

وفى البرنامج التالى نرسم كعكة باستخدام دائرة ثابتة نصف قطرها 100 ومركزها منتصف الشاشة وذلك بتحريك مركز دائرة أخرى على محيط هذه الدائرة بمقدار 360 درجة (٢ ط) .

وكما ذكرنا من قبل فإنه قد تم استخدام حلقة تكرارية لتأخير الرسم ويصل عداد هذه الحلقة إلى 10000 فلعلك ترغب فى تقليل هذا العداد بما يتناسب مع سرعة الكمبيوتر حتى لا يكون الرسم بطيئاً .



شكل (١١)

كما نرى أيضاً في البرنامج أن مركز الدائرة المتحركة قد تم تحديده كالآتي :

$$\begin{aligned} x &= \text{HalfWidth} + r * \cos(k); \\ y &= \text{HalfHeight} + r * \sin(k); \end{aligned}$$

شكل (١٢)

أى أن الإحداثي x (أو الإحداثي y) مكون من نصف عرض الشاشة (أو ارتفاعها) مضافاً إليه إزاحة قدرها :

$$\begin{aligned} &r * \cos(K) \\ &(\text{أو } r * \sin(K)) \end{aligned}$$

وتكون الإزاحة سالبة أو موجبة بحسب جيب الزاوية أو جيب تمام الزاوية .


```

/* Program 3-4.cpp */
// Downut
#include <graphics.h>
#include <conio.h>
#include <math.h>
#define PI 3.14159
main()
{
    int driver,mode;
    int HalfHeight, HalfWidth;
    double x, y, r, k, Angle, Incr;
    long loop;
    Angle=2*PI;
    Incr=2*PI/360;
    r=100;
    driver=DETECT;
    initgraph(&driver, &mode, "D:/TC/BGI");
    HalfHeight = getmaxy()/2;
    HalfWidth = getmaxx()/2;
    for (k=0; k<=Angle-Incr; k=k+Incr*5) {
        x=HalfWidth+r*cos(k);          // shifting X
        y=HalfHeight+r*sin(k);          // shifting Y
        circle(x,y,50);
        for (loop=1; loop<=10000; loop++);
    }
    getch();
    return (0);
}

```

شكل (١٣)

البرنامج الرابع

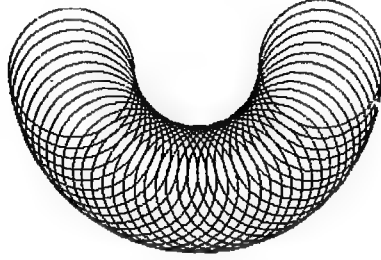
وكما نرى فى البرنامج فإنه قد تم تحديد المسافة الزاوية التى تقطعها الدائرة المتحركة فى رحلتها بالمقدار "2*PI" وكذلك تم تحديد عدد الدوائر المرسومة بالمتغير بخطوة الحلقة التكرارية :

$$K = K + \text{Incr} * 5$$

والمتغير Incr قيمته درجة واحدة بالتقدير الستينى ، فإذا أردت زيادة عدد الدوائر المرسومة يمكنك تصغير الخطوة بتغيير العدد 5 إلى عدد أصغر .

تدريب (٣-٣)

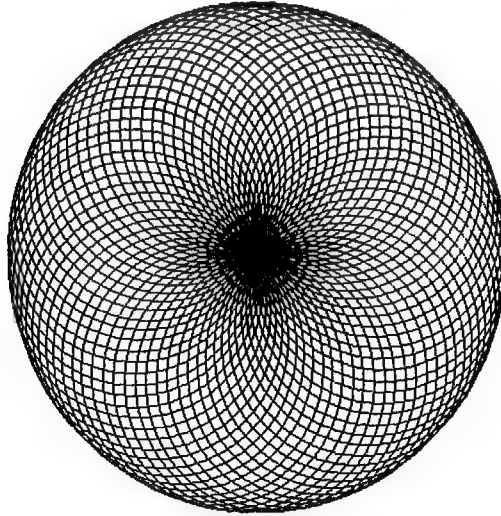
أجر التعديلات اللازمة على البرنامج الرابع لكي يؤدي إلى رسم الأشكال التالية :



شكل (١٤)

انظر الحل في البرنامج DRL3-3A

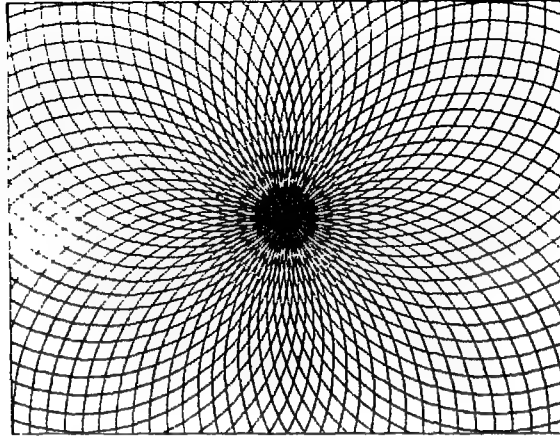
نبرة : تتحرك الدائرة زاوية مقدارها نصف دائرة (١٨٠ درجة)



شكل (١٥)

انظر الحل بالبرنامج DRL3-3B

نبرة : يتساوى هنا نصف قطر الدائرة المتحركة والدائرة الثابتة .



شكل (١٦)

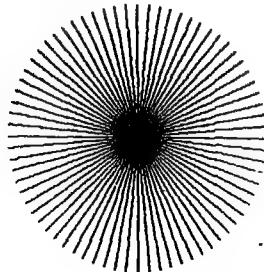
انظر الحل بالبرنامج DRL3-3C

نقطة: يصل هنا نصف قطر أى من الدائرتين إلى نصف ارتفاع الشاشة .

moveto (٣ - ٣) الخطوط

يمكننا بتحريك الخطوط المستقيمة الحصول على أشكال متعددة . فعلى سبيل المثال يمكننا تمثيل نصف قطر الدائرة أثناء حركته الزاوية وذلك بمد خط مستقيم يصل ما بين نقطة ثابتة (المفترض أنها مركز الدائرة) ونقطة أخرى تتحرك على محيط الدائرة ، انظر الشكل التالى :

تنفيذ البرنامج الخامس



شكل (١٧)

وفيما يلي نص البرنامج المؤدى إلى الشكل السابق وقد استخدمنا فيه الدالة `lineto` وقد كان من الجائز استخدام أى دالة أخرى من دوائر رسم المستقيمات مع إجراء التعديلات اللازمة . كما ظهرت دالة جديدة في هذا البرنامج لنقل "الموقع الحالي" الرسم إلى موقع جديد . هذه هي الدالة `moveto` التى تأخذ الصورة العامة :

```
#include <graphics.h>
void far moveto(int x, int y);
```

شكل (١٨)

وتستخدم الدالة `moveto` سابقة لرسم الخط دائماً حتى يبدأ الخط من نفس النقطة في كل مرة يُرسم فيها .

```
/* Program 3-5.cpp */
// Rotating line
#include <graphics.h>
#include <conio.h>
#include <math.h>
#define PI 3.14159
main()
{
    int driver,mode;
    int HalfHeight, HalfWidth;
    double x, y, r, k, Angle, Incr;
    long loop;
    Angle = 2*PI;
    Incr = 2*PI/360;
    r = 150;           // length of the line
    driver=DETECT;
    initgraph(&driver, &mode, "D:/TC/BGI");
    HalfHeight = getmaxy()/2;
    HalfWidth = getmaxx()/2;
    for (k = 0; k <= Angle-Incr; k = k+Incr*5) {
        x = HalfWidth + r*cos(k);           // shifting X
```

```

y = HalfHeight + r*sin(k); // shifting y
moveto(HalfWidth,HalfHeight);
lineto(x,y);
for {loop = 1; loop <= 10000; loop++};
}
getch();
return (0);
}

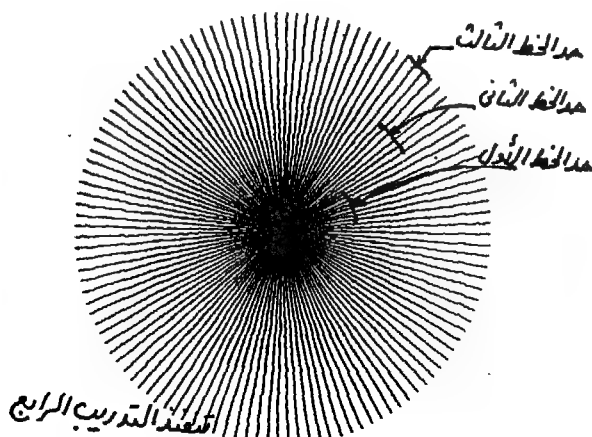
```

شكل (١٩)

تدريب (٣ - ٤)

في الشكل التالي نرى رسماً مشابهاً إلى حد كبير للرسم الناتج من البرنامج السابق (3-5.cpp) ومع ذلك فهناك فوارق ، حيث يتكون هذا الرسم من ثلاثة خطوط دائرة ذات أطوال مختلفة وألوان مختلفة . وعندما يبدأ تنفيذ البرنامج نرى الخط الدائر الأول (أصغرهم طولاً) . فإذا انتهى الخط الأول من دورته الكاملة بدأ الخط الثاني ويلون جديد ، وعندما يكمل رحلته الدائرية يتغير اللون مرة أخرى ويبدأ الخط الثالث .

أجر ما يلزم من تعديلات على البرنامج السابق للحصول على هذا الشكل .

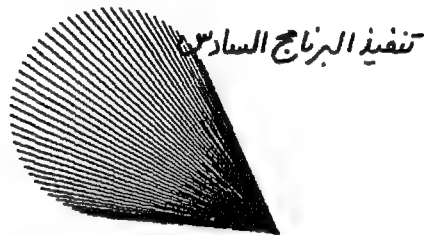


شكل (٢٠)

مخروط باستخدام الخط الدائر

فى البرنامج السابق كان طول الخط المستقيم ثابتاً حيث يصل دائماً بين نقطة ثابتة (تمثل مركز الدائرة) ونقطة أخرى على محيط الدائرة .

لو أننا أجرينا تعديلاً على البرنامج السابق وذلك بإزاحة الدائرة فى الاتجاهين الأفقى والرأسى بحيث تقع النقطة خارج الدائرة ؛ فى هذه الحالة نحصل على الشكل المخروطى الموضح بالشكل .



شكل (٢١)

تنفيذ البرنامج السادس ،

```
/* Program 3-6.cpp */
// Cone
#include <graphics.h>
#include <conio.h>
#include <math.h>
#define PI 3.14159
main()
{
    int driver, mode;
    int HalfHeight, HalfWidth;
    double x, y, r, k, Angle, Incr;
    long loop;
    Angle = 2*PI;
    Incr = 2*PI/360;
    r = 100;           // radius of the circle
    driver=DETECT;
    initgraph(&driver, &mode, "D:/TC/BGI");
```

```

HalfHeight = getmaxy()/2;
HalfWidth = getmaxx()/2;
for (k = 0; k <= Angle-Incr; k = k+Incr*5) {
    x = HalfWidth/2.0 + r*cos(k);           // shifting x
    y = HalfHeight/2.0 + r*sin(k);         // shifting y
    moveto(HalfWidth,HalfHeight);
    lineto(x,y);
    for (loop = 1; loop <= 10000; loop++);
}
getch();
return (0);
}

```

شكل (٢٢)

البرنامج السادس

قوقعة باستخدام الخط الدائر

هناك تعديل آخر يمكن إجراؤه على البرنامج الخامس فيتحول شكل المروحة الدائرية إلى شكل قوقعة . إن هذا يتم بإزاحة الدائرة بحيث تظل النقطة واقعة بداخلها ولكنها أقرب ما يكون إلى المحيط . والأشكال التالية توضح الشكل الناتج والبرنامج السابع .

```

/* Program 3-7.cpp */
// Shell
#include <graphics.h>
#include <conio.h>
#include <math.h>
#define PI 3.14159
main()
{
    int driver, mode;
    int HalfHeight, HalfWidth;
    double x, y, r, k, Angle, Incr;
    long loop;
    Angle = 2*PI;
    Incr = 2*PI/360;
    r = 130;           // radius
    driver=DETECT;
    initgraph(&driver, &mode, "D:/TC/BGI");
}

```

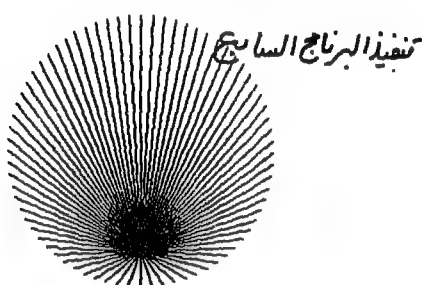
```

HalfHeight = getmaxy()/2;
HalfWidth = getmaxx()/2;
for (k = 0; k <= Angle-Incr; k = k+Incr*5) {
    x = HalfWidth + r*cos(k);           // shifting X
    y = 0.6 * HalfHeight + r*sin(k);    // shifting Y
    moveto(HalfWidth,HalfHeight);
    lineto(x,y);
    for (loop = 1; loop <= 10000; loop++);
}
getch();
return (0);
}

```

شكل (٢٣)

البرنامج السابق

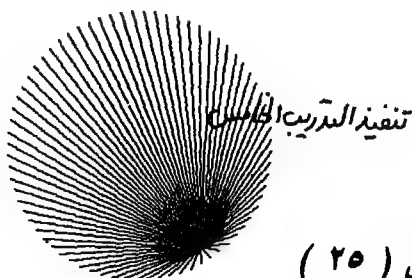


تنفيذ البرنامج السابق

شكل (٢٤)

تدريب (٣ - ٥)

أجر التعديلات اللازمة على البرنامج السابق حتى ينتج الرسم
الموضح بالشكل :



تنفيذ التدريب الخامس

شكل (٢٥)

(٣ - ٤) الرسم بالبكسلات *putpixel*

فى إمكانك رسم نقطة واحدة (بكسل) عند إحداثى معين باستخدام الدالة *putpixel* التى تأخذ الصورة الآتية :

```
#include <graphics.h>
void far putpixel(int x, int y, int color);
```

شكل (٢٦)

وكما نرى من عينة الدالة أن البارامترات تشمل الإحداثيات واللون .
وتستخدم الدالة *putpixel* فى التطبيقات التى لا تسعفنا فيها دوال رسم
الأشكال الجاهزة والتى يلزم رسمها نقطة نقطة .

موجة جيبيه

وفى البرنامج التالى نعرض مثلاً لهذه التطبيقات حيث نرسم موجة جيبيه
(sine wave) .

ومعادلة الموجة الجيبية بصفة عامة هى :

$$y = A \sin(x);$$

حيث A أقصى سعة للموجة الجيبية .

x الزاوية بالتقدير النصف قطرى (radians) .

ويجوز إزاحة المحاور بإضافة أو طرح مقدار ثابت من قيمة y كالتالى :

$$y = Y + A \sin(x);$$

وفى البرنامج التالى تم توقيع المنحنى الجيبى نقطة بنقطة خلال زاوية قدرها

360 درجة وقد قمنا بتحويل الزاوية من التقدير الستيني إلى التقدير نصف القطرى بداخل الحلقة التكرارية .

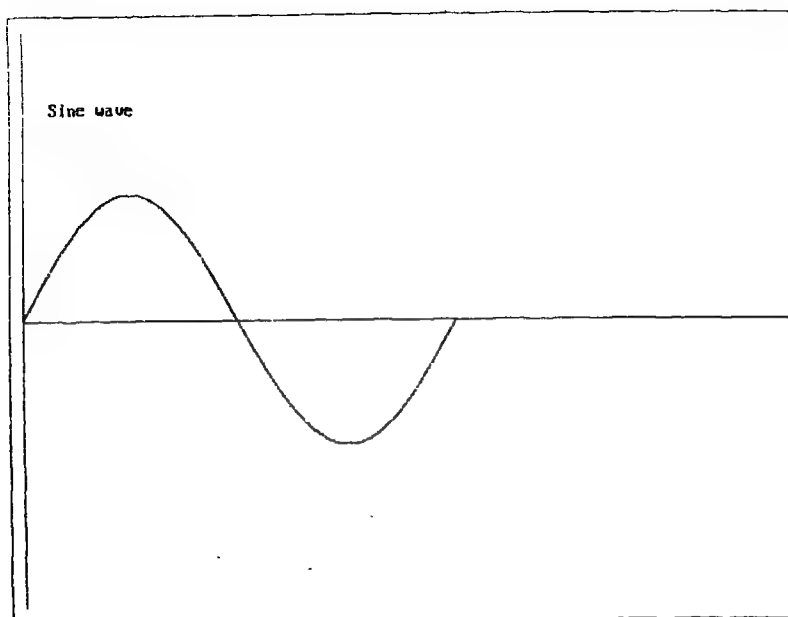
```

/* Program 3-8.cpp */
#include <graphics.h>
#include <conio.h>
#include <math.h>
#include <iostream.h>
#define PI 3.14159
main()
{
    int driver, mode;
    int HalfHeight;
    double x, y, Amp, w;
    long loop;
    driver=DETECT;
    initgraph(&driver, &mode, "D:/TC/BGI");
    HalfHeight = getmaxy()/2;
    line(0,10,0,getmaxy()-10); // Y axis
    line(0,HalfHeight,getmaxx(),HalfHeight); // X axis
    Amp = 100; // Max. Amplitude
    for (x=0; x <= 360; x=x+0.1) {
        w=x*(2*PI/360); // Angular frequency
        y=HalfHeight - Amp*sin(w);
        putpixel(x,y,YELLOW);
    }
    gotoxy(5,5); cout << "Sine wave";
    getch();
    closegraph();
    return (0);
}

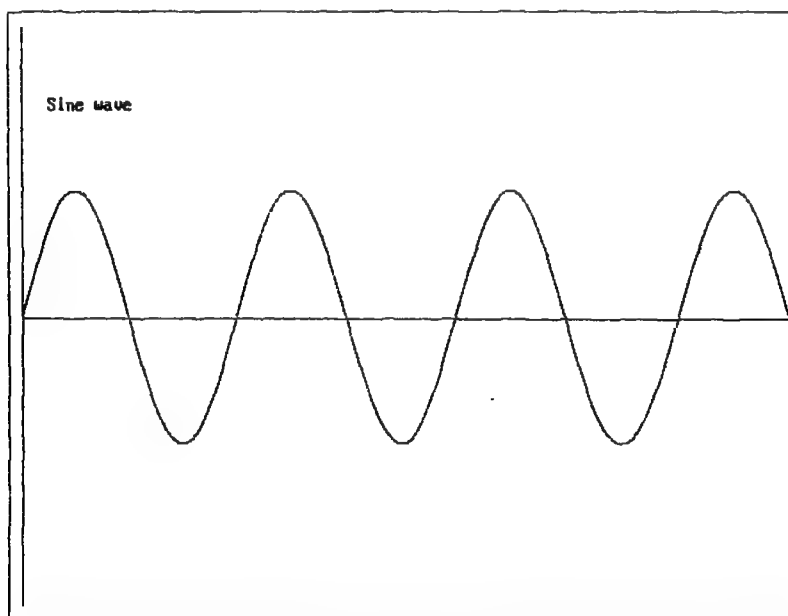
```

شكل (٢٧)

ويرسم هذا البرنامج المحاور x,y علاوة على الموجة الموضحة بالشكل التالى وهى مكونة من ذبذبة واحدة . ويعبر المتغير "w" فى البرنامج عن التردد الزاوى ؛ فإذا أردت الحصول على أكثر من ذبذبة فى الشكل (أى زيادة تردد الموجة) فيمكنك ضرب المتغير "w" فى عدد ثابت فتحصل على شكل مشابه لما فى الشكل رقم (٢٩) .

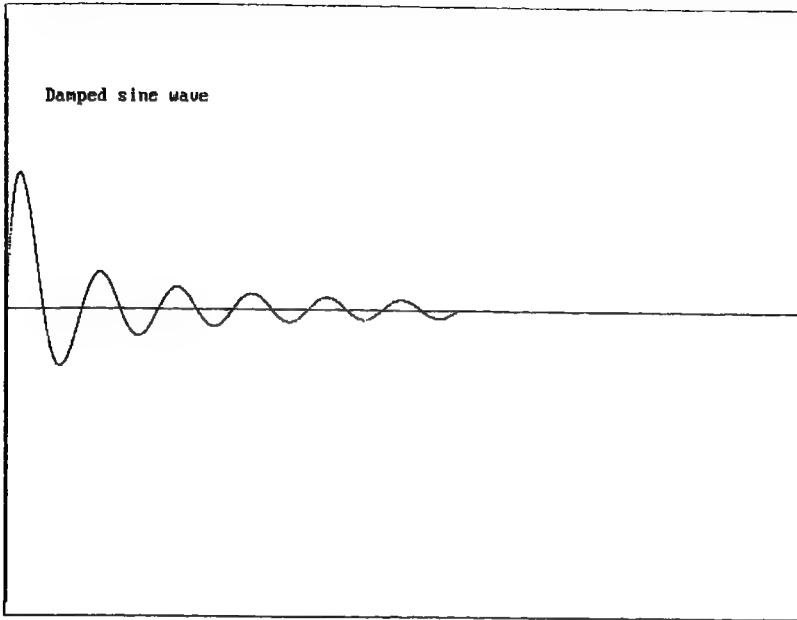


شكل (٢٨)



شكل (٢٩)

موجة جيبية مخمدة *damped sine wave*



شكل (٣٠)

يمثل الشكل ذبذبة مخمدة (Damped oscillations) تتناقص سعتها تدريجياً مع الزمن (أو مع الزاوية) . ويمكنك الحصول على هذه الذبذبة بجعل سعة الموجة الجيبية دالة في المتغير w أو المتغير x .
والعلاقة المستخدمة لرسم هذا الشكل هي :

$$\text{Amp} = 250 / (w+1);$$

ولإضافة العدد 1 إلى المقام هنا يهدف إلى تجنب القسمة على صفر . وهذا هو نص البرنامج الكامل :

```

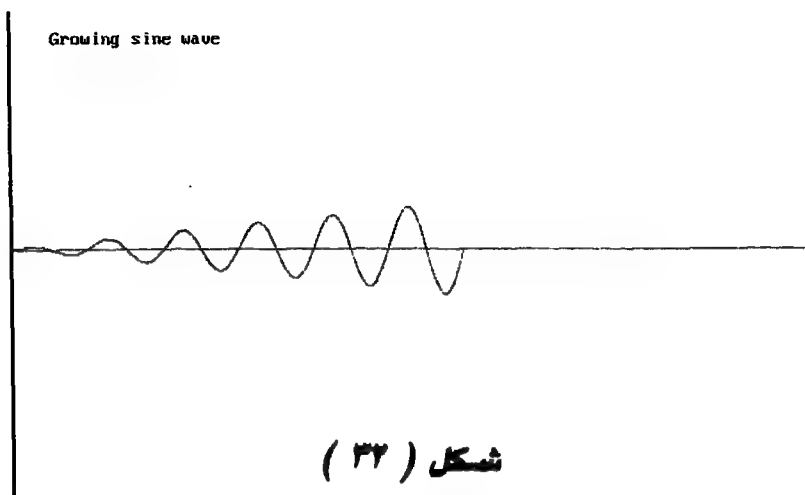
/* Program 3-9.cpp */
#include <graphics.h>
#include <conio.h>
#include <math.h>
#include <iostream.h>
#define PI 3.14159
main()
{
    int driver, mode;
    int HalfHeight;
    double x, y, Amp, w;
    long loop;
    driver=DETECT;
    initgraph(&driver, &mode, "D:/TC/BGI");
    HalfHeight = getmaxy()/2;
    line(0,10,0,getmaxy()-10);           // Y axis
    line(0,HalfHeight,getmaxx(),HalfHeight); // X axis
    for (x=0; x <= 360; x+=0.1) {
        w=6*x*(2*PI/360);           // Angular frequency
        Amp =250/(w+1);             // Max. Amplitude
        y=HalfHeight - Amp*sin(w);
        putpixel(x,y,YELLOW);
    }
    gotoxy(5,5); cout << "Damped sine wave";
    getch();
    closegraph();
    return (0);
}

```

شكل (٣١)
البرنامج التاسع

تدريب (٣ - ٦)

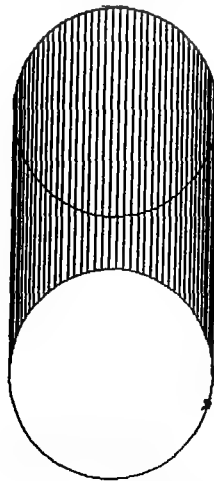
أجر التعديلات اللازمة على البرنامج التاسع حتى تكون النغمة
نامية أى تزداد سعتها بزيادة الزاوية كما فى الشكل التالى :



٣ - ٤) الأشكال المجسمة

اسطوانة مجسمة

فى هذا البرنامج نرسم الشكل الاسطوانى الموضح بالرسم وهو عبارة عن دائرتين تصل بينهما مجموعة من الخطوط المستقيمة .



شكل (٣٣)

والفكرة التي يعتمد عليها هذا الشكل هي كيفية تحديد مواقع بدايات ونهايات الخطوط المستقيمة التي تصل بين الدائرتين . وكما نرى بالشكل فإن جميع النقط التي يبدأ منها المستقيم (أو ينتهي إليها) تقع على محيط نصف دائرة أى أنه يمكن تمثيل الإحداثى الأفقى للنقطة بالمعادلة :

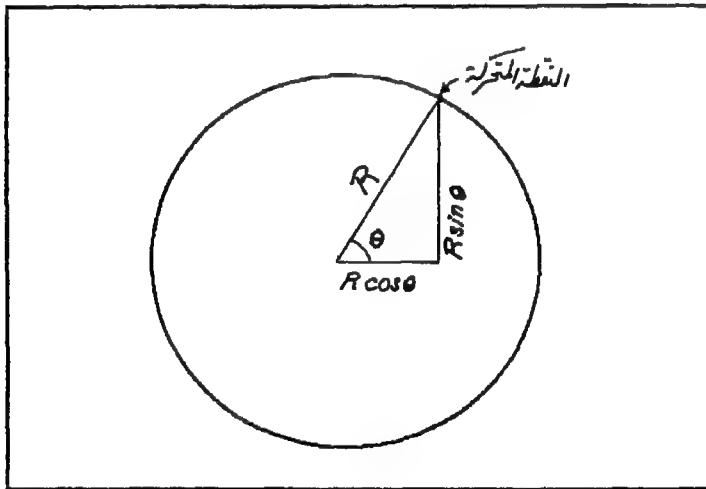
$$x = R \cos(\Theta)$$

حيث R هو نصف قطر الدائرة .

Θ هي الزاوية التي يصنعها نصف القطر مع المحور الأفقى . كذلك يمكن تمثيل الإحداثى الرأسى بالمعادلة :

$$y = R \sin(\Theta)$$

انظر الشكل التالى :



شكل (٣٤)

فلنلق نظرة أولى على البرنامج والتنفيذ ثم نلتقى حول المناقشة .

```

/* Program 3-10.cpp */
// Cylinder
#include <graphics.h>
#include <conio.h>
#include <math.h>
#define PI 3.14159
main()
{
    int driver, mode;
    double x1, y1, y2, r;
    long loop;
    driver = DETECT;
    x1 = 200;
    y1 = 100;
    y2 = 350;
    r = 100;
    initgraph(&driver, &mode, "D:/TC/BGI");
    setcolor(YELLOW);

    circle(x1,y1,r);
    circle(x1,y2,r);

    for(int fi=0; fi<=180; fi+=4) {

        line(x1+r*cos(fi*PI/180), y1-r*sin(fi*PI/180),
            x1+r*cos(fi*PI/180), y2-r*sin(fi*PI/180));

        for(long i=1; i<=100000; i++);
    }
    getch();
    return(0);
}

```

شكل (٣٥)

مناقشة البرنامج

- [١] يبدأ البرنامج بتحديد قيمة مجموعة من المتغيرات وهي :
- x_1 بعد مركز الاسطوانة عن حافة الشاشة اليسرى .
 - y_1 بعد الدائرة الأولى عن حافة الشاشة العليا .

y_2 بعد الدائرة الثانية عن حافة الشاشة العليا .

r نصف قطر الدائرة .

[٢] يتم بعد ذلك رسم الدائرتين العليا والسفلى باستخدام اللون الأصفر .

[٣] يتم بعد ذلك رسم الخطوط من خلال حلقة تكرارية تعتمد على الزاوية "fi" وذلك بتغيير قيمة الزاوية من صفر إلى 180 درجة ، مع استخدام خطوة للتغيير قدرها ٤ درجات .

أما الإحداثي الأفقي لأي نقطة واقعة على سطح الدائرة العليا فهو يتحدد بجمع المركبة $R \cos(fi)$ على البعد x_1 (ويجوز طرح المركبة من البعد x_1 بهذا يحدد اتجاه بدء العملية : من اليسار إلى اليمين أو بالعكس) .

```
for(int fi=0; fi<=180; fi+=4) {
    line(x1+r*cos(fi*PI/180), y1-r*sin(fi*PI/180),
        x1+r*cos(fi*PI/180), y2-r*sin(fi*PI/180));
}
```

إحداثيات النقطة العليا
إحداثيات النقطة السفلى

شكل (٣٦)

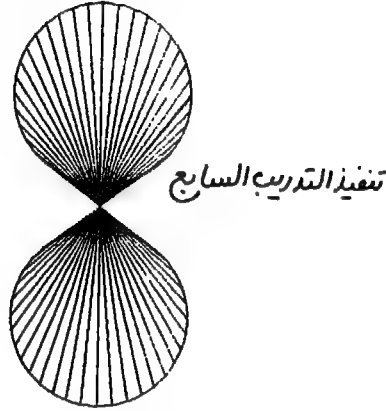
أما الإحداثي الرأسى فهو يتحدد بطرح المركبة $R \sin(fi)$ من البعد الرأسى y_1 أو y_2 (ويجوز أيضاً تغيير الإشارة هنا) .

[٤] كما نلاحظ وجود حلقة تأخير تمكّنك من مشاهدة عملية الرسم بالتصوير البطيء ونذكرك بضرورة تقليل حلقة التأخير إلى رقم معقول مثل 1000 إذا كنت تستخدم كومبيوتراً ذا سرعة بطيئة .

تدريب (٣-٧)

نتج الشكل الموضح بعد من تغيير ما فى البرنامج السابق بحيث أن الاسطوانة قد تحولت إلى هذا الشكل

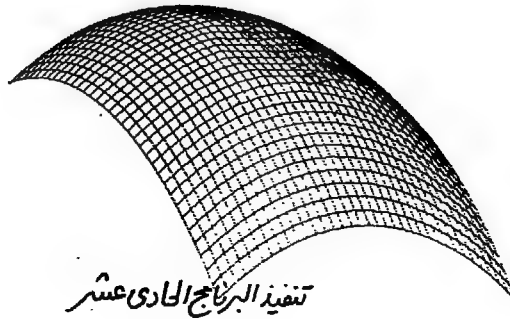
المروحي الذي يمكن أن تحصل عليه (فى الواقع) لو أنك أمسكت باسطوانة ورقية وأثرت عليها بعزم التواء وذلك بإدارة قمتها وقاعدتها فى اتجاهين مختلفين .
أجر التعديل اللازم للحصول على هذا الشكل .



شكل (٣٧)

سطح مجسم

يوضح الشكل التالى جزءاً من سطح دائرى مرسوماً نقطة بنقطة باستخدام الدالة putpixel .



شكل (٣٨)

والبرنامج التالي يوضح منطق رسم السطح الدائري وهو يعتمد أساساً على المعادلة :

$$Z = x^2 + y^2$$

أما الثابت A فهو يتحكم في مساحة السطح ومقدار الإزاحة عن حافة الشاشة اليسرى ، ويتحكم الثابت B في مقدار التقوس .

```
/* Program 3-11.cpp */
#include <graphics.h>
#include <conio.h>
#include <math.h>
#define PI 3.14159
#define A 170
#define B 350
void plotit(void);
int driver, mode;
double x, y, z;
main()
{
    driver=DETECT;
    initgraph(&driver, &mode, "D:/TC/BGI");
    for (x=-A; x<=A; x=x+10)
        for (y=-A; y<=A; y++) plotit();
    for (y=-A; y<=A; y=y+10)
        for (x=-A; x<=A; x++) plotit();
    getch();
    return (0);
}
//
void plotit(void)
{
    double fi=PI/4;
    z=(x*x + y*y) / B;
    putpixel(A + x + (A+y) * cos(fi),
            z + (A+y) * sin(fi),
            2);
}
```

شكل (٣٩)

(٣ - ٦) مواصفات الخطوط

إن استخدام الدالة line أو مثيلاتها من دوال الرسم تؤدي إلى إظهار الخط بمواصفات سابقة التعريف . ومع ذلك فإنه يمكن تغيير مواصفات الخط المرسوم باستخدام الدالة setlinestyle .

ومواصفات الخط المرسوم تشمل الآتي :

● شكل الخط (متصل ، متقطع ، إلى آخره) .

● سمك الخط (سميك ، معتاد) .

والدالة تأخذ الصورة العامة الآتية :

```
void far setlinestyle(int linestyle,
                     unsigned upattern,
                     int thickness);
```

نوع الخط →
نوع السبكة →
السمك →

شكل (٤٠)

ويستخدم البارامتر الأول لتحديد نوع الخط وفقاً لثوابت الماكرو الموضحة بالجدول التالي :

المعنى	القيمة العددية	اسم الماكرو
خط معتاد	0	SOLID_LINE
خط من النقاط	1	DOTTED_LINE
خط من النقاط والشرط	2	CENTER_LINE
خط من الشرط	3	DASHED_LINE
خط من تصميم البرمج	4	USERBIT_LINE

شكل (٤١)

تحديد شكل الخطوط

أما البارامتر الثاني للدالة فهو يختص بتعريف شبكة النقاط التي يتكون منها الخط وسوف نمنح هذا البارامتر القيمة "صفر" في الوقت الحالي .
أما البارامتر الثالث فهو يحدد سمك الخط وفقاً للثوابت الموضحة بالجدول التالي :

اسم الماكرو	القيمة العددية	سمك الخط مقدراً بالبكسلات
NORM_WIDTH	1	1 pixel (خط معتاد)
THICK_WIDTH	3	3 pixels (خط سميك)

شكل (٤٢)

تحديد سمك الخطوط

ويؤثر البارامتر الثالث (سمك الخط) على الخطوط والدوائر (وسائر الأشكال التي ستعرض لها فيما بعد) .
والجدول التالي يوضح بعض نماذج الخطوط ذات السمك العادي وذات السمك الكبير (٣ بكسلات) .

	NORM_WIDTH السمك المعتاد	THICK_WIDTH السمك الكبير
خط متصل SOLID_LINE	—————	—————
خط نقط DOTTED_LINE
خط نقط وشرط CENTER_LINE	-----	-----
خط شرط DASHED_LINE	-----	-----

شكل (٤٣)

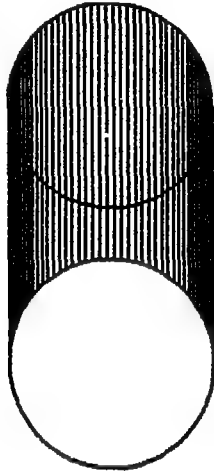
نماذج لأشكال الخطوط

وحتى نجرب استخدام هذه الماكروا ، فلنبدأ بإضافة الدالة `setlinestyle` إلى برنامج الاسطوانة (البرنامج العاشر) وذلك باستخدام

البارامترات الآتية :

SOLID - LINE	● نوع الخط
THICK - LINE	● سمك الخط
صفر	● شبكة النقط

وبذلك نحصل على الشكل التالى حيث نرى الخطوط والدوائر مرسومة بالسمك الكبير .



شكل (٤٤)

وفيما يلى الصورة المعدلة للبرنامج ونلاحظ فيه أن الدالة المسفولة عن سمك الخطوط قد جاءت تالية للدخول فى نسق الرسم :

```
setlinestyle(SOLID_LINE,
              0,
              THICK_WIDTH);
```

شكل (٤٥)

وهذا هو نص البرنامج :

```

/* Program 3-12.cpp */
// Cylinder
#include <graphics.h>
#include <conio.h>
#include <math.h>
#define PI 3.14159
main()
{
    int driver, mode;
    double x1, y1, y2, r;
    long loop;
    driver = DETECT;
    x1 = 200;
    y1 = 100;
    y2 = 350;
    r = 100;
    initgraph(&driver, &mode, "D:/TC/BGI");
    setcolor(YELLOW);
    setlinestyle(SOLID_LINE, 0, THICK_WIDTH);

    circle(x1, y1, r);
    circle(x1, y2, r);

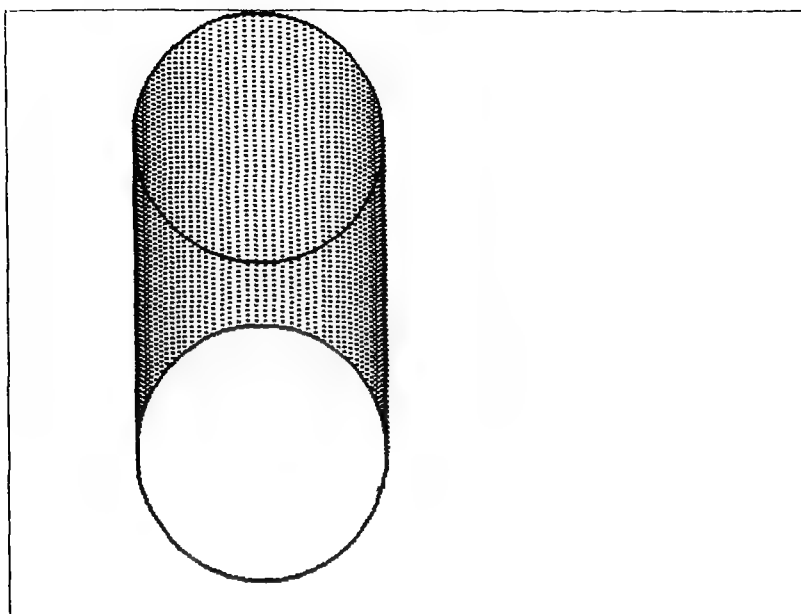
    for(int fi=0; fi<=180; fi+=4) {
        line(x1+r*cos(fi*PI/180), y1-r*sin(fi*PI/180),
            x1+r*cos(fi*PI/180), y2-r*sin(fi*PI/180));
        for(long i=1; i<=100000; i++);
    }
    getch();
    return(0);
}

```

شكل (٤٦)

تدريب (٣ - ٨)

أجر التعديل اللازم على البرنامج السابق لإنتاج الرسم الموضح
بالشكل التالي :

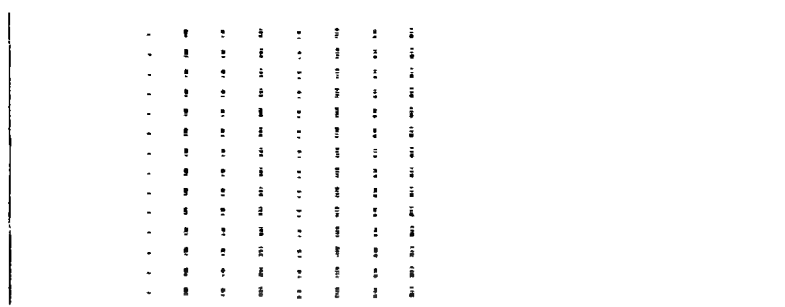


شكل (٤٧)

أنواع الخطوط المبتكرة (User – defined line styles)

يمكنك تفصيل شكل الخطوط المطلوبة حسب الطلب وذلك باختيار البارامتر الأول بالرقم 4 أو بالماكرو USERBIT – LINE وهذا يؤدي إلى إمكانية تعريف الشبكة (pattern) التي يتكون منها الخط . والشبكة عبارة عن مجموعة من الأرقام الثنائية مكتوبة في 16 "بت" . فإذا كانت الشبكة تحتوي على الرقم صفر (أى أن جميع البتات تحتوي على صفر) كان الخط غير مرئي . أما إذا كانت الشبكة تحتوي على الرقم 1 في جميع البتات (وهذا يكافئ العدد السداسى عشر FFFF) فإن هذا يعطى خطاً سميكاً . وما بين الصفر والعدد FFFF تتنوع أشكال الخطوط .

والشكل التالى يعرض بعض نماذج الخطوط المبتكرة التى يمكن الحصول عليها باستخدام الدالة setlinestyle وهى مرسومة بطريقة رأسية .



شكل (٤٨)

ويجوز التعبير عن العدد الممثل للشبكة بأى نظام عددى مع استخدام الشفرة المناسبة لكل نظام . وفى المثال الآتى نستخدم العدد FFFF للـ الشبكة بالرقم 1 :

```
setlinestyle(USERBIT_LINE,0xFFFF,THICK_WIDTH);
```

شكل (٤٩)

أما فى المثال التالى فنستخدم الرقم العشرى 256 للـ الشبكة :

```
setlinestyle(USERBIT_LINE,256,NORM_WIDTH);
```

شكل (٥٠)

كذلك يمكنك إصدار مجموعة كبيرة من الخطوط باستخدام حلقة تكرارية لاختيار منها ما تشاء لاستخداماتك . والمثال التالى يرسم لنا مجموعة الخطوط الموضحة بالشكل (٤٨) :

```
for (int i=16; i<=256;i=i+31) {
    setlinestyle(USERBIT_LINE,i,THICK_WIDTH);
    line(x1+i, y1,x1+i, y2);
}
```

شكل (٥١)

وبطبيعة الحال فإن هذه الحلقة التكرارية لابد أن تدخل ضمن هيكل برنامج متكامل يتم فيه تعريف الأبعاد x_1 ، y_1 ، y_2 .

الموجز

[١] كانت هذه هي رحلتنا الخاطفة مع النقط والخطوط والدوائر وقد تعرفنا خلالها بدوال جديدة ومهارات جديدة ، مثل دالة رسم بكسلة واحدة على الشاشة ودالة نقل الموقع الحالي لنقطة الرسم إلى إحداثي معين . كما عرفنا كيفية التحكم في شكل وسمك الخطوط المرسومة بالدوال المختلفة سواء كانت مواصفات الخطوط مبنية في اللغة أو مبتكرة من عندنا .

[٢] فيما يلي ملخص بالدوال التي التقينا بها في هذا الباب :

دوال التعرف على أبعاد الشاشة

```
#include <graphics.h>
int far getmaxx(void);
int far getmaxy(void);
```

دالة التحرك إلى موقع معين

```
#include <graphics.h>
void far moveto(int x, int y);
```

دالة رسم نقطة (بكسلة)

```
#include <graphics.h>
void far putpixel(int x, int y, int color);
```

دالة التحكم في خصائص الخطوط

```
#include <graphics.h>
void far setlinestyle(int linestyle,
                     unsigned upattern,
                     int thickness);
```

الباب الرابع

دوال الرسم والظلاء



مفتّح

فى هذا الباب تستكشف أغوار الوصلة البينية BGI فننتعرف بما تجود به من دوال للرسم والطلاء . فهناك دوال كثيرة لرسم الأشكال الجاهزة بدلاً من بنائها من المبادئ الأولية .

كما نتعرض فى هذا الباب إلى دوال التحكم فى المساحات الداخلية للأشكال بطلائها باللون المطلوب أو بالشكل الزخرفى المطلوب الذى سنصطلح على تسميته بالشبكة . وعلى وجود شبكات كثيرة ذات أشكال مختلفة مبنية فى اللغة نفسها فيمكنك مع ذلك ابتكار ما تشاء من أشكال الشبكة لاستخدامها فى الطلاء .

وفى النهاية سوف نطبق كثيراً من المهارات التى خبرناها فى هذا الباب على موضع الرسم "بالفراكتلات" (Fractals) وهو أسلوب رياضى جديد لرسم المشاهد الطبيعية المتفرّدة مثل السحاب وقمم الجبال وخط الساحل المتعرج .

(٤ - ١) رسم مستطيل rectangle

تستخدم الدالة rectangle كما هو واضح من اسمها لرسم مستطيل بمعلومية النقط الممثلة لأركانها الأربعة ، وتأخذ الدالة صورة العينة الآتية :

```
#include <graphics.h>
void far rectangle(int left, int top,
                  int right, int bottom);
```

شكل (١)

حيث يمثل "اليسار" بعد الضلع الأيسر للمستطيل عن حافة الشاشة اليسرى ، وتمثل "القمة" بعد الضلع العلوي للمستطيل عن حافة الشاشة العليا . كذلك فإن "اليمن" يمثل بعد الضلع الأيمن عن حافة الشاشة اليسرى (الإحداثي الأفقي) ، ويمثل "القاع" بعد الضلع السفلي عن حافة الشاشة العليا (الإحداثي الرأسي) . وفي البرنامج التالي نبدأ برسم مستطيل باستخدام الأبعاد :

10,100,100,300

ثم تظهر رسالة على الشاشة تطلب منك الضغط على أى زر فإذا ضغطت على أحد الأزرار رأيت سلسلة متتابعة من المستطيلات التى لها نفس المساحة تولد من المستطيل الأول . والبرنامج تطبيق مباشر على استخدام الدالة ولا يحتاج إلى تعليق .

```
/* Program 4-1.cpp */
// Rectangles
#include <graphics.h>
#include <conio.h>
```

```
#include <iostream.h>
main()
{
    int driver, mode;
    int left=10, top=100;
    int right=100, bottom=300;
    long loop;
    driver = DETECT;
    initgraph(&driver, &mode, "D:/TC/BGI");
    setcolor(YELLOW);
    // المستطيل الأول
    rectangle(left, top, right, bottom);
    cout << "Press a key..";
    getch();
    // عائلة المستطيلات
    for (int r=10; r<=150; r=r+10) {
        rectangle(left+r, top+r, right+r, bottom+r);
        for (loop=1; loop<=10000; loop++);
    }
    getch();
    closegraph();
    return (0);
}
```

شكل (٢)



شكل (٣)



(٤ - ٢) الأشكال المضلعة drawpoly

تستخدم الدالة drawpoly لرسم الأشكال المضلعة بمعلومية مجموعة النقاط التي يصل بينها الشكل المضلع . وعينة هذه الدالة هي :

```
#include <graphics.h>
void far drawpoly(int numpoints,
                  int far *polypoints);
```

شكل (٤)

وكما نرى في عينة الدالة فهي تستخدم بارامترين :

- numpoints ويمثل عدد أركان المضلع (أو عدد أضلاعه) .
- polypoints اسم المصفوفة العددية التي تخزن فيها إحداثيات النقاط الممثلة للأركان .

معنى ذلك أننا لو أردنا رسم مضلع مكون من خمسة أضلاع فعلينا باستخدام القيمة 5 للبارامتر الأول وتخزين خمسة أزواج من الأعداد في مصفوفة تمثل إحداثيات النقاط الخمس .

والشكل المضلع الناتج بهذه الطريقة لن يكون مغلقاً أى أن الضلع الخامس لن يظهر في الرسم ؛ ولذلك فإذا أردنا إكمال الشكل المضلع لابد من إضافة نقطة سادسة لها إحداثيات النقطة الأولى ومعنى هذا أن يزيد البارامتر الأول بمقدار 1 (فيصبح 6 بدلاً من 5) ويزيد عدد عناصر المصفوفة بمقدار 2 (إحداثيات النقطة الجديدة) .

انظر هذا البرنامج الذي يرسم مضلعاً مكوناً من خمسة أضلاع يصل ما بين النقاط الآتية :

(100,100),(200,300),(250,190),
(300,300),(140,250)

```

/* Program 4-2.cpp */
// Polygons
#include <graphics.h>
#include <conio.h>
//
#define NoOfPoints 5
#define NoOfCoord NoOfPoints*2 .
//
main()
{
    int driver, mode;
    int PointArray[NoOfCoord+2] = {100,100, 200,130
                                   ,250,190, 300,300
                                   ,140,250, 100,100};

    driver = DETECT;
    initgraph(&driver, &mode, "D:/TC/BGI");
    setcolor(YELLOW);
    // Set line style:
    setlinestyle(SOLID_LINE,0,THICK_WIDTH);
    // Draw the polygon:
    drawpoly(NoOfPoints+1,PointArray);
    //
    getch();
    closegraph();
    return(0);
}

```

شكل (٥)

ونلاحظ في هذا البرنامج إعلان عدد النقط الممثلة لأركان الشكل المضلع في صورة ماكرو :

```

#define NoOfPoints 5

```

ثم جاء الإعلان عن أبعاد مصفوفة الإحداثيات مساوياً ضعف عدد النقط :

```

#define NoOfCoord NoOfPoints*2

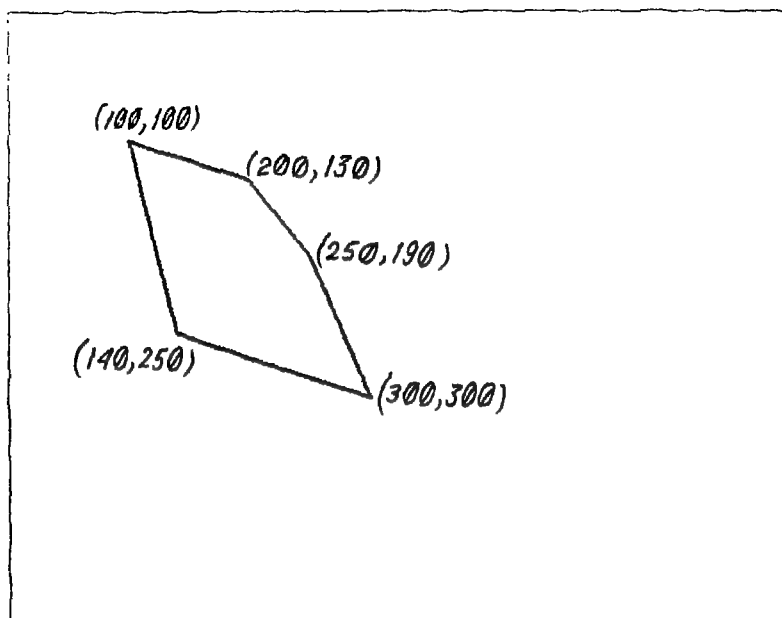
```

كما نلاحظ أن بعد المصفوفة المستخدم بالفعل في البرنامج يزيد بمقدار 2 عن البعد المُعلن وذلك حتى يتم إقفال الشكل بإضافة النقطة الأولى كعنصر سادس بالمصفوفة .

كما أنه عند استخدام الدالة drawpoly تم إضافة 1 إلى عدد النقط لنفس السبب :


```
drawpoly(NoOfPoints+1,PointArray);
```

والشكل التالى يوضح تنفيذ البرنامج :



شكل (٦)

ملء مساحة الشكل المضلع fillpoly

يمكنك باستخدام الدالة fillpoly أن تحصل على شكل مضلع مصمت حيث يتم تلوينه باللون سابق التعريف للرسم .

وتأخذ هذه الدالة نفس البارامترات التى تستخدمها دالة الرسم drawpoly كالتالى :

```
#include <graphics.h>
void far fillpoly(int numpoints,
                  int far *polypoints);
```

شكل (٧)

ولكى تجرب هذه الدالة أضف العبارة الآتية إلى البرنامج السابق بدلاً من عبارة رسم المضلع "drawpoly" كالتالى :

```
fillpoly(NoOfPoints+1,PointArray);
```

وسوف تلاحظ عند تنفيذ البرنامج أن المضلع المرسوم له إطار أصفر اللون (وهو اللون الناتج من استخدام العبارة `setcolor`) أما مساحة المضلع فتظهر بلون أبيض . والسبب في ذلك أن اللون المخصص لملء المساحات يتم التحكم فيه باستخدام دالة خاصة هي الدالة "`setfillstyle`" فإذا لم نستخدم هذه الدالة لتحديد لون الملء فإن اللون سابق التعريف (الأبيض) هو الذى يستخدم فى الملء .

التحكم فى ألوان وأشكال المساحة المصمتة `setfillstyle`

يتم التحكم فى لون المساحات المصمتة وكذلك فى شكل الطلاء بالدالة الآتية :

```
void far setfillstyle(int pattern, int color);
```

شكل (٨)

وتأخذ هذه الدالة بارامترين هما :

◎ اللون (Color) :

وهو يأخذ أحد القيم المعروفة للون . ويؤثر بارامتر اللون على المساحات المصمتة فقط ولكنه لا يغير من اللون الحالى للرسم أو الخلفية .

◎ شكل الطلاء (pattern) :

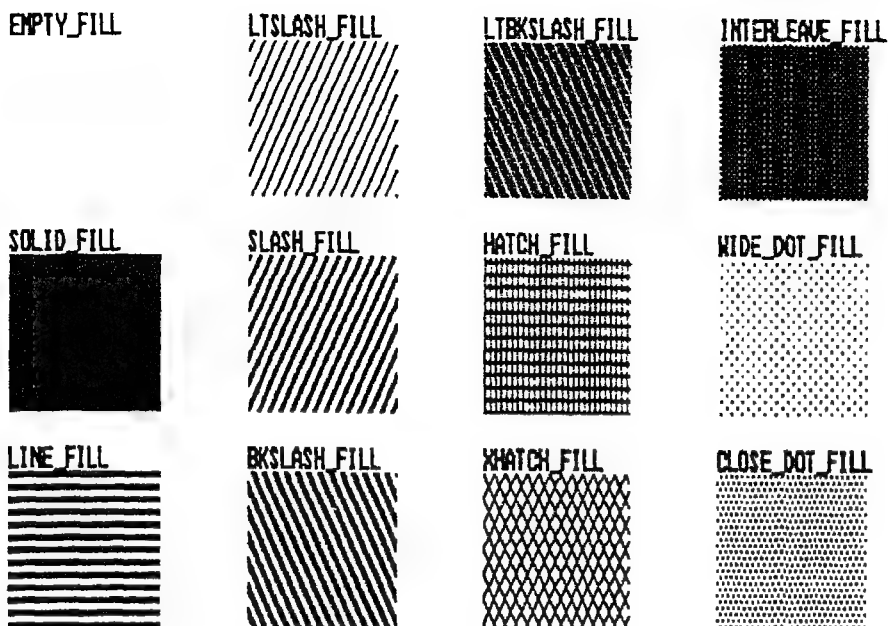
ويمكن تحديد هذا البارامتر بأى عدد ما بين الصفر ، 12 أو بأسماء الماكرو الموضحة بالجدول التالى . ويؤدى اختيار أى رقم (أو ماكرو) إلى تحديد الشكل الزخرفى المستخدم للطلاء والذى نستخدم عليه باسم "شبكة الطلاء" .

والأشكال المختلفة لشبكة الطلاء موضحة بالشكل التالى للجدول مع بيان اسم الماكرو المناظر لكل شبكة أعلى الرسم .

القيمة العددية	اسم الماكرو
0	EMPTY_FILL (مناظر لونه الخلفية)
1	SOLID_FILL
2	LINE_FILL
3	LTSLASH_FILL
4	SLASH_FILL
5	BKSLASH_FILL
6	LTBKSLASH_FILL
7	HATCH_FILL
8	XHATCH_FILL
9	INTERLEAVE_FILL
10	WIDE_DOT_FILL
11	CLOSE_DOT_FILL
12	USER_FILL (شبكة من ابتكار المبرمج)

شكل (٩)
شبكات الطلاء





شكل (١٠)

أشكال شبكات الطلاء المستخدمة في الملء

والشكل التالي يوضح برنامج رسم المضلع (البرنامج الثاني) في صورة معدلة حيث استخدمنا فيه الدالة `setfillstyle` لاختيار شبكة الطلاء رقم 7 (`HATCH-FILL`) وكذلك اللون الأزرق للطلاء الداخلي . ثم استخدمنا الدالة `fillpoly` لرسم المضلع المصمت .

وبل البرنامج الشكل المتوقع الحصول عليه من تنفيذ البرنامج :

```
/* Program 4-3.cpp */
// Filled Polygons
#include <graphics.h>
#include <conio.h>
//
#define NoOfPoints 5
#define NoOfCoord NoOfPoints*2
//
main()
{
```

```

int driver, mode;
int PointArray[NoOfCoord+2] = {100,100, 200,130
                                ,250,190, 300,300
                                ,140,250, 100,100};

driver = DETECT;
initgraph(&driver, &mode, "D:/TC/BGI");
setcolor(YELLOW);
// Set line style:
setlinestyle(SOLID_LINE,0,THICK_WIDTH);
// Draw the polygon and fill with blue HATCH pattern:
setfillstyle(HATCH_FILL, BLUE);
fillpoly(NoOfPoints+1,PointArray);
//
getch();
closegraph();
return(0);
}

```

شكل (١١)



شكل (١٢)



تدريب (٤ - ١)

لعله من المفيد في هذه المرحلة أن تستكشف بنفسك أشكال شبكات
الطلاء بعرضها على الشاشة .
أجر التعديل اللازم على البرنامج الثالث حتى تحصل على جميع
أشكال شبكات الطلاء المتاحة من خلال حلقة تكرارية .

ولنا جولة قادمة مع شبكات الطلاء سوف نتحدث فيها عن شبكات الطلاء
المبتكرة .

(٤ - ٣) رسم الأقواس الدائرية arc

تستخدم الدالة arc لرسم الأقواس وهي تأخذ الصورة العامة :

```
#include <graphics.h>
void far arc(int x, int y, int stangle,
             int endangle, int radius);
```

زاوية البداية ↗
↖ زاوية النهاية ↗ نصف القطر ↗
شكل (١٣)

حيث : x,y مركز الدائرة (لاحظ أن القوس جزء من دائرة) .
stangle زاوية البداية بالدرجات .
endangle زاوية النهاية بالدرجات .
radius نصف القطر مقاساً بالبكسل .

وعندما يرسم القوس فإن اتجاه الدوران (من زاوية البداية إلى زاوية النهاية
يكون عكس اتجاه عقارب الساعة) .

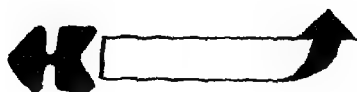
ومعنى ذلك أن دالة القوس arc يجوز أن تستخدم لرسم جزء من دائرة أو دائرة كاملة . فلو أنك اخترت زاوية البداية "صفر" وزاوية النهاية "360" حصلت على الدائرة الكاملة . أما أى عدد أقل من 360 فيعطى قوساً .

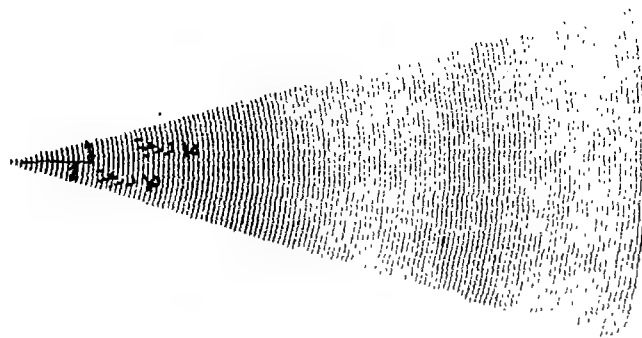
ومن الجائز أن تكون الزاوية سالبة وهذا يعنى أن الزاوية مُقاسة ضد عقارب الساعة .

وفي البرنامج التالى نرسم الإشعاع الصادر عن أحد الفئارات وهو عبارة عن قوس ما بين $(+15)$ درجة و (-15) درجة و يتزايد نصف قطره مع المسافة .

```
/* Program 4-4.cpp */
// Arcs
#include <graphics.h>
#include <conio.h>
main()
{
    int driver, mode;
    driver = DETECT;
    initgraph(&driver, &mode, "D:/TC/BGI");
    setcolor(YELLOW);
    for(int radius=0; radius<=getmaxx(); radius+=5)
        arc(0, getmaxy()/2,
            -15, 15,
            radius);
    getch();
    closegraph();
    return(0);
}
```

شكل (١٤)





شكل (١٥)

منشأ إحداثيات القوس	arccoordstype
	getarccoods

فى بعض الأحيان قد يتطلب الأمر توصل بداية القوس أو نهايته بخط مستقيم كما هو الحال عند رسم نصف دائرة أو ربع دائرة .

ويستلزم الأمر فى هذه الحالة حساب إحداثيات البداية والنهاية وقد تكون هذه العملية شاقة فى معظم الأحيان .

بدلاً من ذلك فإن الوصلة البينية BGI تمدنا بالمنشأ الذى يحتوى على جميع إحداثيات القوس المرسوم مؤخراً . وهذا هو تعريف المنشأ: arccoordstype

```
struct arccoordstype {
    int x, y;    المركز
    int xstart, ystart;
    int xend, yend;
};
```

شكل (١٦)

ويمكنك استخدام أعضاء المنشأ مباشرة فى برنامجك بمجرد الإعلان

عن متغير من النمط arccoordstype واستخدام الدالة getarccoords التي ترجع قيم هذه الأعضاء . وعينة هذه الدالة كالآتي :

```
void far getarccoords(
    struct arccoordstype far *arccoords);
```

شكل (١٧)

وكما نرى فى عينة الدالة أنها تستخدم دليلاً واحداً عبارة عن مؤشر إلى المنشأ .

فلو أنك فى برنامجك أعلنت عن متغير مثل P من النمط arccoordstype فإنك بمجرد استدعاء هذه الدالة يمكنك الحصول على البيانات الآتية :

- إحداثيات المركز P.x, P.y
- إحداثيات بداية القوس P.xstart, P.ystart
- إحداثيات نهاية القوس P.xend, P.yend

وفى هذا البرنامج نرسم نصف دائرة ثم نوصل بدايتها بنهايتها بخط مستقيم باستخدام البيانات الموجودة فى المنشأ arccoordstype كما يطبع البرنامج على الشاشة بيانات المنشأ إذا أردت الاطلاع عليها .

```
/* Program 4-5.cpp */
// Arcs
#include <graphics.h>
#include <conio.h>
#include <iostream.h>
main()
{
    int driver, mode;
    driver = DETECT;
    arccoordstype P;
    initgraph(&driver, &mode, "D:/TC/BGI");
    int radius = getmaxy()/2;
    setcolor(YELLOW);
    arc(getmaxx()/2, getmaxy()/1.5,
        0, 180, radius);
    getarccoords(&P);
```

```

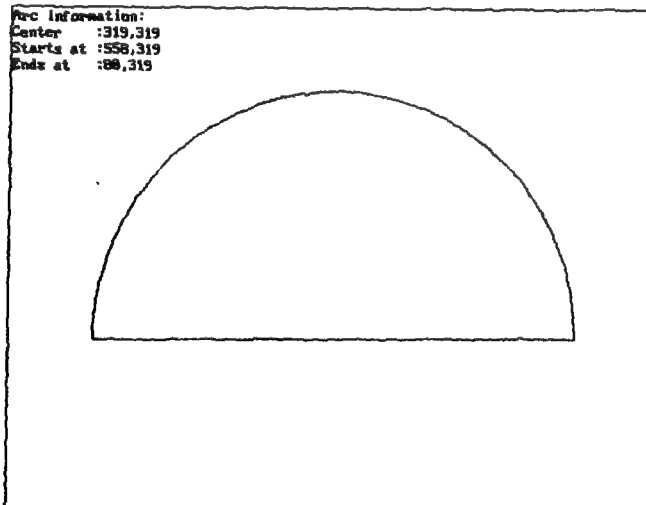
line(P.xstart, P.ystart,
     P.xend, P.yend);
cout << "Arc information:" << endl
     << "Center      :" << P.x
     << "," << P.y << endl
     << "Starts at  :" << P.xstart
     << "," << P.ystart << endl
     << "Ends at    :" << P.xend
     << "," << P.yend << endl;
getch();
closegraph();
return(0);
}

```

شكل (١٨)

ملاحظة

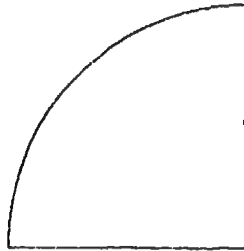
إذا كنت تستخدم لغة سي فإنه يلزم استخدام كلمة `struct` عند إعلان المنشأ . هذا علاوة على التعليقات الأخرى المذكورة في الملحق (د).



شكل (١٩)

تدريب (٤ - ٢)

اكتب البرنامج الذى يرسم الشكل التالى :



شكل (٢٠)

(٤ - ٤) القطع الناقص ellipse

تستخدم الدالة ellipse لرسم القطع الناقص ، وهى تأخذ الصورة العامة :

```
#include <graphics.h>
void far ellipse(int x, int y,
                 int stangle, int endangle,
                 int xradius, int yradius);
```

شكل (٢١)

وكما نرى من عينة الدالة أنها تشبه دالة رسم القوس arc إلى حد كبير حيث تستخدم المركز (x,y) وزاوية بداية (stangle) وزاوية نهاية (endangle) ، كلاهما يقاس بالدرجات فى عكس عقارب الساعة .

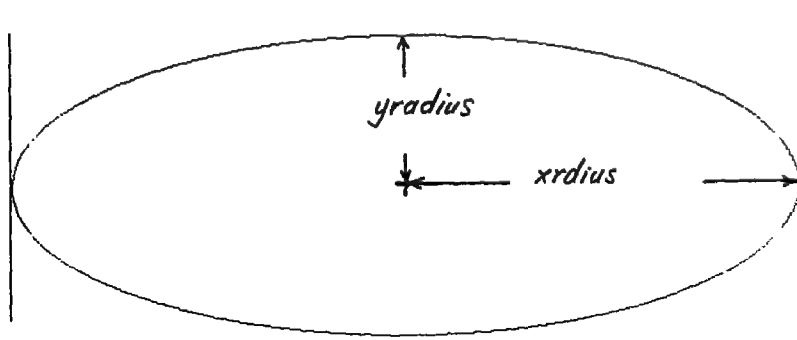
أما الخاصية التى ينفرد بها القطع الناقص فهى وجود نصفى قطر ، واحد فى الاتجاه الرأسى (yradius) وآخر فى الاتجاه الأفقى (xradius) .

ويجوز استخدام هذه الدالة في رسم الأقواس (لا سيما الأقواس غير الدائرية) وذلك إذا كان الفرق بين زاوية البداية وزاوية النهاية أقل من 360 درجة .

وفي البرنامج التالي نعرض مثلاً توضيحياً لاستخدام الدالة `ellipse` .

```
/* Program 4-6.cpp */
// Ellipses
#include <graphics.h>
#include <conio.h>
main()
{
    int driver, mode;
    driver = DETECT;
    initgraph(&driver, &mode, "D:/TC/BGI");
    int HalfWidth = getmaxx()/2;
    int HalfHeight = getmaxy()/2;
    int xradius = getmaxx()/2;
    int yradius = getmaxy()/4;
    setcolor(YELLOW);
    ellipse(HalfWidth, HalfHeight,
           0, 360,
           xradius, yradius);
    getch();
    closegraph();
    return(0);
}
```

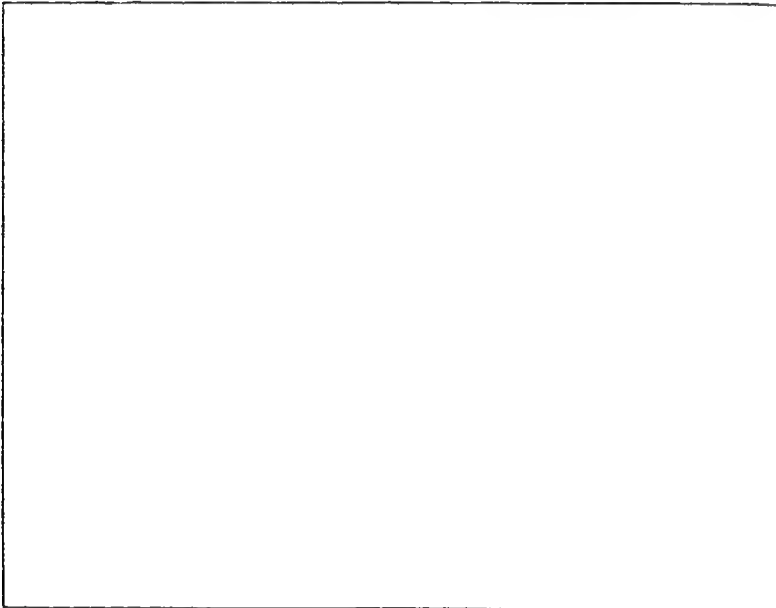
شكل (٢٢)



شكل (٢٣)

تدريب (٤ - ٣)

أجر التعديلات اللازمة على البرنامج السابق لكي يرسم الشكل
الموضح بعد علماً بأن عرض الشكل يساوى ضعف طوله . لاحظ
أيضاً استخدام الخط السميك .



شكل (٢٤)

`fillellipse`

ملء القطع الناقص

`getmaxcolor`

تستخدم الدالة `fillellipse` لرسم قطع ناقص مع طلائه من الداخل باللون
المطلوب والشبكة المطلوبة .

والدالة `fillellipse` تأخذ الصورة العامة الآتية :

```
#include <graphics.h>
void far fillellipse(int x, int y,
                    int xradius, int yradius);
```

شكل (٢٥)

وتشبه الدالة دالة رسم القطع الناقص في جميع الأوجه فيما عدا أنها لا تتضمن زاويتي البدء والنهاية لأنها ترسم قطعاً دائماً ولا تستخدم في رسم الأقواس .

وفي المثال التالي نرسم قطعاً ناقصاً باستخدام هذه الدالة مع ملئه بالشبكات المختلفة من خلال حلقة تكرارية تبدأ من الشبكة "EMPTY-FILL" وهي الشبكة رقم "صفر" وتنتهي عند الشبكة "CLOSE-DOT-FILL" وهي الشبكة رقم 11 [انظر الجدول شكل (٩) والرسم شكل (١٠)] .

أما اللون المستخدم فلم يتم تحديده صراحة بل استخدمنا بدلاً من ذلك الدالة getmaxcolor وهي ترجع أكبر رقم من أرقام الألوان بالنسبة للطور المستخدم في الرسم وهذا يجعل البرنامج عاماً ويصلح للتشغيل مع مختلف كروت الرسم . والصيغة العامة للدالة getmaxcolor هي :

```
int far getmaxcolor(void);
```

شكل (٢٦)

ويمكنك أن تطبع القيمة المرتجعة من هذه الدالة للتحقق من رقم اللون المستخدم .

```
/* Program 4-7.cpp */
// Filling an ellipse
#include <graphics.h>
#include <conio.h>
#include <iostream.h>
main()
{
```

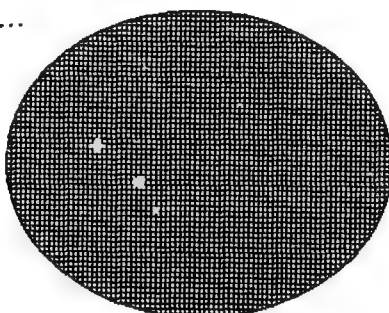
```

int mode, driver = DETECT;
int HalfWidth, HalfHeight;
int xradius, yradius;
initgraph(&driver, &mode, "d:/tc/bgi");
HalfWidth = getmaxx()/2;
HalfHeight = getmaxy()/2;
xradius = HalfWidth/2;
yradius = HalfHeight/2;
cout << endl << "Press any key...";
for (int i = EMPTY_FILL; i <= CLOSE_DOT_FILL; i++) {
    setfillstyle(i, getmaxcolor());
    fillellipse(HalfWidth, HalfHeight,
                xradius, yradius);
    getch();
}
closegraph();
return 0;
}

```

شكل (٢٧)

Press any key...



أحد مشاهد البرنامج السابع

شكل (٢٨)

القطاع sector

أما الدالة sector فهي تستخدم لرسم قطاع (شريحة) قطع ناقص مع طلائه بالشبكة سابقة التعريف أو التي يتم تحديدها بالدالة setfillstyle . وعينة الدالة كالآتي :

```
#include <graphics.h>
void far sector(int x, int y,
               int stangle, int endangle,
               int xradius, int yradius);
```

شكل (٢٩)

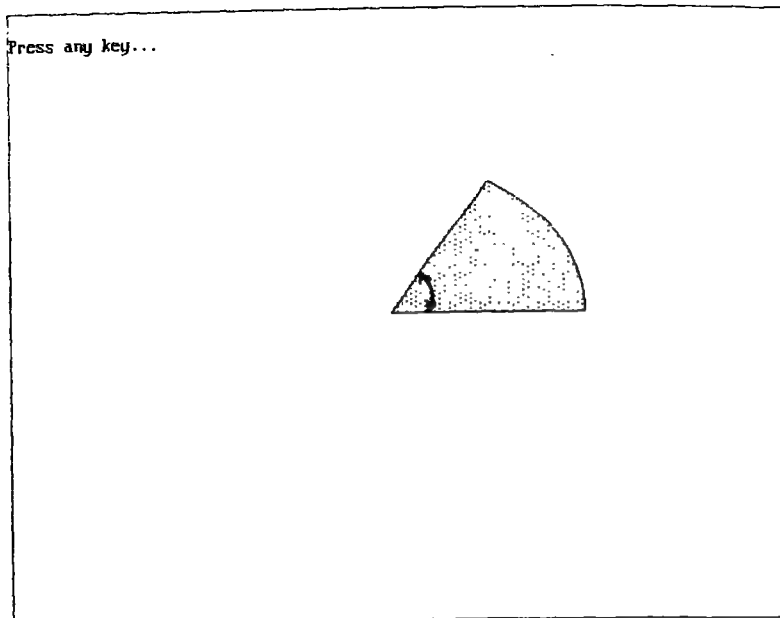
وكما نرى أن عينة الدالة sector تماثل تماماً عينة الدالة ellipse فهي تستخدم لنفس الهدف فيما عدا أن القطاع يظهر دائماً مطلياً ولا يظهر فارغاً .

وفي البرنامج التالى نكرر منطق البرنامج السابع مع استبدال الدالة fillellipse بالدالة sector مع تحديد زاوية القطاع من صفر إلى 60 درجة .

```
/* Program 4-8.cpp */
// Sector
#include <graphics.h>
#include <conio.h>
#include <iostream.h>
main()
{
    int mode, driver = DETECT;
    int HalfWidth, HalfHeight;
    int xradius, yradius;
    int stangle = 0, endangle = 60;
    initgraph(&driver, &mode, "d:/tc/bgi");
    HalfWidth = getmaxx()/2;
    HalfHeight = getmaxy()/2;
    xradius = HalfWidth/2;
    yradius = HalfHeight/2;
    cout << endl << "Press any key...";
    for (int i = EMPTY_FILL; i <= CLOSE_DOT_FILL; i++) {
        setfillstyle(i, getmaxcolor());
        sector(HalfWidth, HalfHeight,
              stangle, endangle,
              xradius, yradius);

        getch();
    }
    closegraph();
    return (0);
}
```

شكل (٣٠)



شكل (٣١)

القطاع الدائري *pieslice*

إن القطاع الذي رسمناه في البرنامج الثامن كان جزءاً من قطع ناقص ويطلق عليه أحياناً شريحة الفطيرة (pie Slice) غير الدائرية أما إذا كان القطاع جزءاً من دائرة فيطلق عليه شريحة الفطيرة عموماً .

وتستخدم الدالة *pieslice* في رسم القطاع الدائري وهي تأخذ الصورة العامة الآتية :

```
#include <graphics.h>
void far pieslice(int x, int y, int stangle,
                  int endangle, int radius);
```

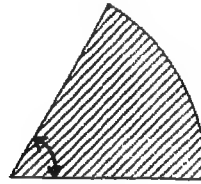
شكل (٣١) ١

وتتشابه بارامترات هذه الدالة مع بارامترات دالة القوس arc تماماً . وهي دائماً تؤدي إلى رسم قطاع مطلي باللون سابق التعريف والشبكة سابقة التعريف ما لم يتم استخدام الدالة "setfillstyle" .

تدريب (٤ - ٤)

اكتب البرنامج الذي يؤدي إلى رسم القطاع الدائري الموضح بعد (بزاوية قدرها ٦٠ درجة) ، على أن يتم تغيير شبكة الطلاء من خلال حلقة تكرارية كلما ضغطت على أى زر :

Press any key...



شكل (٣٢)

(٥ - ٤) القضبان المستوية والمجسمة

إن القضبان المستوية ما هي إلا المربعات والمستطيلات (أى المضلعات الرباعية) ولكن الفرق بينها وبين المضلعات الرباعية المرسومة بالدالة

rectangle أنها أشكال ذات طلاء داخلي وبلا برواز خارجي .
وتستخدم الدالة bar لهذا الغرض وهي تأخذ الصورة الآتية :

```
#include <graphics.h>
void far bar(int left, int top,
             int right, int bottom);
```

شكل (٣٣)

حيث :

- top ، left هي إحداثيات الركن الأيسر العلوى للشكل الرباعي .
- bottom ، right هي إحداثيات الركن الأيمن السفلى للشكل الرباعي .

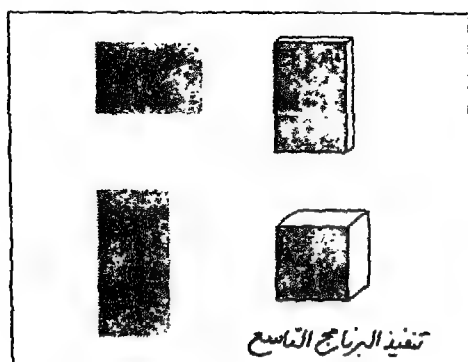
أما الدالة bar3d فتستخدم لرسم القضبان المجسمة المصمتة ولذلك فهي تستخدم ستة أدلة ، الأربعة الأولى منها تماثل أدلة الدالة bar (انظر عينة الدالة بالشكل التالى) :

```
#include <graphics.h>
void far bar3d(int left, int top,
               int right, int bottom,
               int depth, int topflag);
```

شكل (٣٣) ١

أما الدليل الخامس هنا فيمثل العمق مقاساً بالبكسل ، وأما الدليل السادس فهو مجرد راية تأخذ القيمة "صفر" فتختفى قمة الشكل أو تأخذ قيمة غير صفرية فتظهر قمة الشكل كالمعتاد .

والشكل التالي يوضح أربعة قضبان اثنان منها (على اليسار) مرسومان بالدالة bar واثنان مرسومان بالدالة bar3d باستخدام أبعاد مختلفة .



شكل (٣٤)

ويوضح الشكل التالي نص البرنامج الذي أدى إلى هذا الرسم .

```
/* Program 4-9.cpp */
// Bar and Bar3D
#include <graphics.h>
#include <conio.h>
main()
{
    int mode, driver = DETECT;
    int HalfWidth, HalfHeight;
    initgraph(&driver, &mode, "d:/tc/bgi");
    HalfWidth = getmaxx()/2;
    HalfHeight = getmaxy()/2;
    setfillstyle(CLOSE_DOT_FILL, getmaxcolor());
    bar(HalfWidth-200, HalfHeight-200,
        HalfWidth-50, HalfHeight-100);
    bar(HalfWidth-200, HalfHeight,
        HalfWidth-100, HalfHeight+200);
    bar3d(HalfWidth+150, HalfHeight-200,
        HalfWidth+50, HalfHeight-50, 10, 1);
    bar3d(HalfWidth+50, HalfHeight+50,
        HalfWidth+150, HalfHeight+150, 30, 1);
    getch();
    closegraph();
    return (0);
}
```

شكل (٣٥)

تدريب (٤ - ٥)

جرب استبدال الرقم 1 فى الدالة bar3d بالرقم 0 وشاهد التغيرات
الحادثة فى الشكل المجسم .

(٤ - ٦) شبكات الطلاء المبتكرة setfillpattern

عرفنا من قبل الدالة setfillstyle وكيفية استخدامها لاختيار أحد شبكات
الطلاء المرقمة من صفر إلى 11 .

يمكنك مع ذلك تعريف شبكة الطلاء الخاصة بك وذلك باستخدام الدالة
الجديدة setfillpattern التى تأخذ الصورة الآتية :

```
#include <graphics.h>
void far setfillpattern(char far *upattern,
                        int color);
```

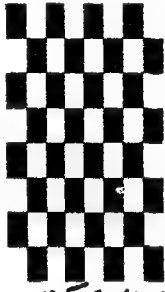
شكل (٣٦)

حيث : upattern مؤشر يشير إلى حيز من الذاكرة قدره ٨ بايت
تختص كل منها بتعريف ثمانية بكسلات فى الشبكة
المبتكرة .

ومعنى ذلك أن كل بكسله يتم تمثيلها بخلية واحدة (بت BIT) قد تأخذ

الرقم 1 أو الرقم 0 . فإذا احتوت البت على الرقم 1 كانت البكسل مضاءة وإذا احتوت على الرقم صفر كانت البكسل مطفأة (لا تظهر على الشاشة) . وعلى سبيل المثال فالمصفوفة الآتية تؤدي إلى تعريف ثمانية "بايت" بحيث تعطى شكل لوحة الشطرنج .

ونرى في الشكل التعريف مكتوباً بالنظام السداسي عشر مثل "0x55" وبجواره الرقم بالنظام الثنائي (كملاحظة بالبرنامج) يليه الرسم المعبر عن شكل البكسلات . والشكل الناتج يمثل وحدة شبكة الطلاء المتكررة التي يمكن أن نملأ بها الأشكال .

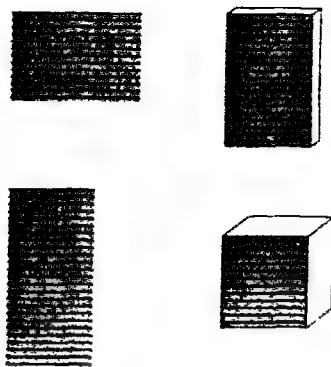
char checkerboard[8] = {				
0xAA,	/*	10101010	=	
0x55,	/*	01010101	=	*/
0xAA,	/*	10101010	=	*/
0x55,	/*	01010101	=	*/
0xAA,	/*	10101010	=	*/
0x55,	/*	01010101	=	*/
0xAA,	/*	10101010	=	*/
0x55	/*	01010101	=	*/
الرقم السداسي عشر : الرقم الثنائي شكل الشبكة				

شكل (٣٧)

تعريف شبكة على شكل لوحة شطرنج

وفي البرنامج التالي نستخدم الدالة **setfillpattern** لتفصيل شبكة طلاء مكونة من خطين بينهما مجموعة من المربعات الصغيرة ثم نستخدم هذه الشبكة في طلاء مجموعة القضبان التي رسمناها في البرنامج التاسع .





شكل (٣٨)

```

/* Program 4-10.cpp */
// User-defined patterns
#include <graphics.h>
#include <conio.h>
main()
{
    char MyPattern[8] = {
        0xFF, /* 11111111 */
        0xAA, /* 10101010 */
        0x00, /* 00000000 */
        0xAA, /* 10101010 */
        0x00, /* 00000000 */
        0xAA, /* 10101010 */
        0x00, /* 00000000 */
        0xFF /* 11111111 */
    };
    int mode, driver = DETECT;
    int HalfWidth, HalfHeight;
    initgraph(&driver, &mode, "d:/tc/bgi");
    HalfWidth = getmaxx()/2;
    HalfHeight = getmaxy()/2;
    setfillpattern(MyPattern, getmaxcolor());
    bar(HalfWidth-200, HalfHeight-200,
        HalfWidth-50, HalfHeight-100);
    bar(HalfWidth-200, HalfHeight,
        HalfWidth-100, HalfHeight+200);
    bar3d(HalfWidth+150, HalfHeight-200,
        HalfWidth+50, HalfHeight-50, 10, 1);
    bar3d(HalfWidth+50, HalfHeight+50,
        HalfWidth+150, HalfHeight+150, 30, 1);
}

```

```

getch();
closegraph();
return (0);
}

```

شكل (٣٩)

ملاحظة

لو عدت إلى الشكل رقم (٩) الذى يوضح أرقام وأسماء شبكات الطلاء المختلفة المستخدمة مع الدالة "setfillstyle" لوجدت أن الرقم 12 يناظر الشبكة USER-FILL وهى الشبكة المبتكرة التى يقوم بتعريفها المبرمج . ومع ذلك فلا تستخدم الدالة setfillstyle لتحديد شبكة الطلاء المبتكرة كما يبدو الأمر لأول وهلة ، فكما عرفنا أن شبكة الطلاء المبتكرة تستلزم استخدام الدالة setfillpattern . أما الشبكة رقم 12 (أو الماكرو USER-FILL) فهو يستخدم مع الدالة :

getfillpattern

التي تستخدم للتعرف على الشبكة الحالية . وسوف نترك استكشاف خصائص هذه الدالة للقارئ .

(٧-٤) طلاء الأشكال بدالة الفيضان floodfill

قد لمسنا حتى الآن أن دوال الطلاء موجهة لطلاء مساحات معينة مثل القضبان والقطاعات والأشكال المضلعة . ولكنك بلا شك سوف تحتاج إلى دالة لطلاء منطقة ما ليس لها شكل محدد سابق التعريف .

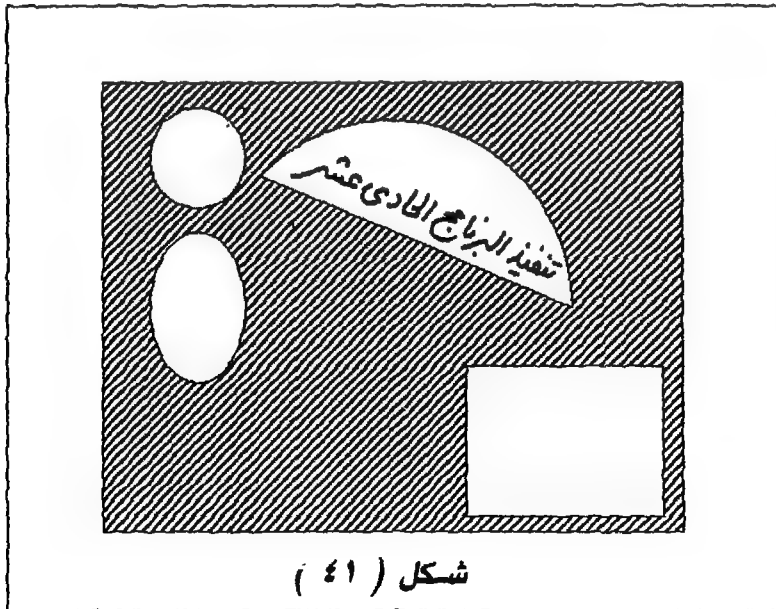
إن الدالة floodfill تستخدم لهذا الغرض وكل ما يلزمك أن تحدد المنطقة المطلوب طلائها بحدود لون معين . وهذه هى عينة الدالة :


```
#include <graphics.h>
void far floodfill(int x, int y,
    int border);
```

شكل (٤٠)

حيث x, y الإحداثيات التي تبدأ منها عملية الطلاء أو بمعنى آخر التي يبدأ منها "الفيضان". أما "border" فيمثل اللون المعبر عن الحد الذي يتوقف عنده الفيضان.

وكما نرى في الشكل التالى أننا قد رسمنا مجموعة من الأشكال الهندسية (دائرة وقطع ناقص وقطاع ومستطيل) وكلها مرسومة باللون الأبيض؛ ثم رسمنا مستطيلاً بنفس اللون يحيط بجميع الأشكال. وقد بدأنا الطلاء من مركز الشاشة تماماً مع استخدام اللون الأبيض كحد للطلاء (border) وبذلك فإننا نرى أن المساحة كلها قد تم طلاؤها فيما عدا الحدود المقفلة ذات اللون الأبيض.



ونلاحظ كذلك في الشكل السابق أننا قد استخدمنا شبكة الطلاء
SLASH-FILL. وبالطبع فإنه يجوز استخدام أى من شبكات الطلاء
وذلك باستدعاء الدالة setfillstyle كما يجوز استخدام أى لون من الألوان
باستخدام الدالة setcolor قبل عملية الطلاء.
وهذا هو البرنامج الذى نتج عنه الرسم السابق.

```
/* Program 4-11.cpp */
// Flood Fill
#include <graphics.h>
#include <conio.h>
main()
{
    arccoordstype P;
    int mode, driver = DETECT;
    int HalfWidth, HalfHeight;
    int xradius, yradius;
    initgraph(&driver, &mode, "d:/tc/bgi");
    HalfWidth = getmaxx()/2;
    HalfHeight = getmaxy()/2;
    setcolor(WHITE);
    arc(HalfWidth, HalfHeight, '
        0,135,150);
    getarccoords(&P);
    line(P.xstart, P.ystart,
        P.xend, P.yend);
    circle(HalfWidth/2, HalfHeight/2,40);
    ellipse(HalfWidth/2, HalfHeight,
        0,360,40,60);
    rectangle(HalfWidth*1.2, HalfHeight*1.2,
        HalfWidth*1.7, HalfHeight*1.7);
    rectangle(HalfWidth/4, HalfHeight/4,
        HalfWidth*1.75, HalfHeight*1.75);
    setfillstyle(SLASH_FILL, YELLOW);
    floodfill(HalfWidth, HalfHeight,WHITE);
    getch();
    closegraph();
    return (0);
}
```

شكل (٤٢)

(٤ - ٨) الرسم باستخدام "الفراكتلات" (Fractals)

هل تأملت يوماً خط الساحل المتعرج حيث يلتقى البحر بالرمال ؟ وهل أسعدك الحظ برحلة إلى ساحل البحر الأحمر لكي تشاهد سلسلة الجبال العجيبة التي تمتد على الساحل من شمال مصر إلى جنوبها لتصنع تحفة فنية رائعة ؟ وهل تأملت في الخط المنكسر الذي تصنعه قمم الجبال المتصلة في الأفق ؟

ولو أنك عقدت مقارنة بين خط الساحل المتعرج في منطقة ما وخط الساحل المتعرج في مناطق أخرى فمن المستحيل أن تجد تطابقاً بين أى جزئين . ومما لا شك فيه أن هناك قانوناً إلهياً وراء هذه المعجزات ولكننا في حدود علمنا نسميه قانون العشوائية التي بها تصطف ملايين النقاط لتصنع هذا الخط المتفرد لسطوح الجبال أو خطوط السواحل .

وقد حاول بعض علماء الرياضيات تصميم نموذج لتمثيل الطبيعة وما تنطوي عليه من تعقيدات فانتهى بهم الأمر إلى ابتكار هندسة "الفراكتلات" (Fractal geometry) .

وتستخدم الفراكتلات في رسم الأشكال الطبيعية المعقدة التي تخضع للعشوائية مثل السواحل والجبال والبحيرات والأشجار والسحب . كما تستخدم الفراكتلات في التصوير بالأقمار الصناعية لاستكمال المعلومات الناقصة في الصور ولا سيما الخرائط .

ولننظر إلى الشكل التالي الذي يمثل مشهداً طبيعياً لجبال البحر الأحمر عند التقائها بصفحة السماء ومياه البحر عند التقائها بالشاطئ . إن التعرجات العشوائية المثلثة لسطح الجبل وتلك المثلثة لخط الساحل مرسومة باستخدام هندسة الفراكتلات ولكننا نلاحظ - رغم العشوائية - أن تعرجات الساحل أكثر نعومة من تعرجات الجبل الحادة . معنى ذلك أن هناك قانوناً يعمل جنباً إلى جنب مع العشوائية .

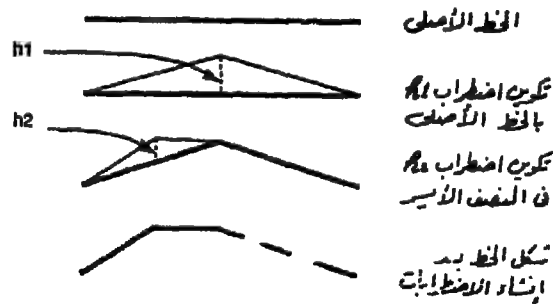


شكل (٤٣)

والمنظر المرسوم ليس موقعاً نقطة بنقطة فهو أساساً خطان مستقيمان : واحد للجبل وواحد للساحل ، وتطبيق مبدأ "الفركتلة" يتعرج كل منهما بالأسلوب المناسب .

ويتميز الرسم بالفراكتلات (بالكمبيوتر) بأن البرنامج الواحد يرسم مشهداً مختلفاً في كل تشغيل بمعنى أن أماكن التعرج وحدة التعرج تختلف في كل مرة برغم من أن الموقع الجغرافي للجبل أو الساحل ثابت دائماً .

ويوضح الشكل التالي خطوات عملية "الفركتلة" التي بها يتحول خط مستقيم إلى خط متعرج طبيعي .



شكل (٤٤)

عملية "الفركتلة" (Fractalization)

وتقوم عملية الفركتلة على مبدأ تكوين مجموعة من الاضطرابات (perturbations) في الخط المستقيم ؛ وذلك بتقسيمه إلى نصفين وإقامة عمود من منتصفه بارتفاع عشوائى قدره "h1" فنحصل على شكل مثلثى . وبحذف قاعدة المثلث والاكتفاء بالضلعين نحصل على الاضطراب الأول كما بالشكل . وبتقسيم كل من الضلعين بنفس الأسلوب وبارتفاعات مختلفة h2 ، h3 .. نحصل على اضطرابات جديدة في الخط المستقيم . وتستمر هذه العملية حتى تتعذر عملية التقسيم أبعد من ذلك .

ويتحكم مقدار الارتفاعات h1 ، h2 ، في حدة الاضطراب الناتج ، ومن البديهي أن مقدار الارتفاع h2 أقل من h1 ومقدار الارتفاع h3 أقل من h2 وهكذا ...

والشكل التالى يوضح البرنامج الذى يرسم منظر الساحل والجبل .

```
/* Program 4-12.cpp */
// Fractals
#include <graphics.h>
#include <conio.h>
#include <math.h>
#include <stdlib.h>
#include <time.h>
//
#define SIZE 1000
#define MAXPERTURB 6
//
double Frac_Array[SIZE];
void Fractalize(int,int,double,double);
void Limits(int,int,int,int,double,double);
void Draw_It(int);
main()
{
    int driver, mode;
    driver=DETECT;
    initgraph(&driver, &mode, "d:/TC/BGI");
```

```
// Initialize random function:
randomize();
// Fractalize the mountain line and draw it:
setcolor(CYAN);
Limits(100,100,MAXPERTURB,0.3,50.0);
Draw_It(100);
// Fill the sky with CYAN color:
setfillstyle(SOLID_FILL,CYAN);
floodfill(1,1,CYAN);
// Fractalize the seashore line and draw it:
setcolor(WHITE);
Limits(150,150,MAXPERTURB,0.9,30.0);
Draw_It(150);
// Fill the sea with BLUE color:
setfillstyle(SOLID_FILL,BLUE);
floodfill(1,getmaxy()-1,WHITE);
// Draw the frame of the screen:
rectangle(0,0,getmaxx(),getmaxy());
// Wrap up:
getch();
closegraph();
return(0);
}
```

شكل (٤٥)

الجزء الأول من البرنامج الثاني عشر

```
// A function to process the limits
// of fractalization:
void Limits(int y1,int y2,
            int MaxPerturb,
            double DecayFctr,double FrScale)
{
    int First, Last;
    double FrRatio, FrDecay;
    First=0;
    Last=(int)pow(2.0,(double)MaxPerturb);
    Frac_Array[First]=y1;
    Frac_Array[Last]=y2;
    FrRatio=1.0/pow(2.0,DecayFctr);
    FrDecay=FrScale*FrRatio;
```

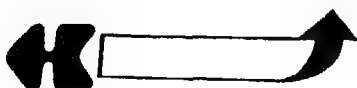
```

    Fractalize(First,Last,FrDecay,FrRatio);
}
//
// The Fractalization process:
void Fractalize(int y1,int y2,
                double FrDecay,double FrRatio)
{
    int Middle;
    double NewFrScale;
    Middle=(y1+y2)/2;
    if(Middle != y1 && Middle !=y2) {
        Frac_Array[Middle]=
            (Frac_Array[y1]+Frac_Array[y2])/2 +
            (double)((random(16)-8))/8.0*FrDecay;
        NewFrScale=FrDecay*FrRatio;
        Fractalize(y1,Middle,NewFrScale,FrRatio);
        Fractalize(Middle,y2,NewFrScale,FrRatio);
    }
}
//
// Drawing Fractalized lines:
void Draw_It(int y)
{
    int xinc, L;
    L= (int)pow(2.0,(double)MAXPERTURB);
    xinc=getmaxx()/L*3/2;
    moveto(0,y);
    for(int i=0, x=0; i<L; i++, x+=xinc)
        lineto(x,(int)Frac_Array[i]);
}

```

شكل (٤٦)

الجزء الثانى والأخير من البرنامج الثانى عشر



مناقشة البرنامج

[١] يبدأ البرنامج يرسم خط الجبل باللون التيركواز (CYAN) (بعد إجراء عملية الفركتلة عليه) ثم يتم طلاء السماء في المنطقة المحصورة ما بين الجبل وأعلى الشاشة باللون التيركواز (اللون التيركواز أقرب ما يكون للون السماوى). أما دالة رسم الجبل فتتم على مرحلتين :

● أولاً : استدعاء الدالة "Limits" التى تقوم باستدعاء الدالة . "Fractalize" لإجراء عملية التقسيم ونشر الاضطرابات المطلوبة .

● ثانياً : استدعاء الدالة "Draw - It" التى ترسم الخط المضطرب من خلال حلقة تكرارية باستخدام دالة الرسم lineto .

ويرسم الجبل على ارتفاع قدره 100 من أعلى الشاشة .

```
// Fractalize the mountain line and draw it:
setcolor(CYAN);
Limits(100,100,MAXPERTURB,0.3,50.0);
Draw_It(100);
```

شكل (٤٧)

[٢] يتم بعد ذلك رسم خط الساحل بنفس الأسلوب على ارتفاع قدره 150 من أعلى الشاشة .

```
// Fractalize the seashore line and draw it:
setcolor(WHITE);
Limits(150,150,MAXPERTURB,0.9,30.0);
Draw_It(150);
```

شكل (٤٨)

[٣] أما عملية الطلاء فتتم باستخدام الدالة floodfill وباستخدام شبكة الطلاء المصنعة SOLID-FILL . وهذه هى دالة طلاء السماء التى تبدأ من الركن (1,1) مع ملاحظة أن نقطة بداية الطلاء لا أهمية لها ، فالمهم أن تبدأ من أى نقطة بداخل المساحة المطلوب طلاؤها لأن الفيضان يتدفق فى جميع الجهات ويتوقف عند اللون المحدد فى الدالة .


```
// Fill the sky with CYAN color:
setfillstyle(SOLID_FILL,CYAN);
floodfill(1,1,CYAN);
```

شكل (٤٩)

[٤] بعد رسم الجبل والساحل وطلاء السماء والبحر بالألوان المناسبة نرسم إطاراً أبيض للشكل كله . ويمكنك الاطلاع على خطوات الرسم وذلك بإضافة الدالة getch في مواضع مختلفة في البرنامج .

[٥] أما عملية الفركتلة فهي تبدأ من الدالة Limits كما ذكرنا . وتستخدم هذه الدالة مجموعة بارامترات .

```
// A function to process the limits
// of fractalization:
void Limits(int y1,int y2,
            int MaxPerturb,
            double DecayFctr,double FrScale)
```

شكل (٤٩) ١

وتمثل البارامترات y1 ، y2 الإحداثي الرأسى لبداية ونهاية الخط المطلوب بث الاضطرابات فيه . وقد استخدمنا الأرقام 100,100 بالنسبة لخط الجبل والأرقام 150,150 بالنسبة لخط الساحل .

ويمثل البارامتر الثالث "Maxperturb" عدد مرات تقسيم الخط الجارى بث الاضطرابات فيه ويأخذ هذا البارامتر القيمة الثابتة 6 التى حددناها فى بداية البرنامج بالماكرو "MAXPERTURB" .

أما البارامتران الرابع والخامس فهما يحددان معاً (من خلال العلاقات الرياضية بالدالة) شكل الخط الناتج بعد عملية الفركتلة . ويسمى البارامتر السادس مقياس الفركتلة "FrScale" لأنه يحدد مقدار الاضطراب اللازم فى كل خطوة ؛ كما يسمى البارامتر السادس بمعامل الاضمحلال "DecayFctr" فهو يودى إلى تقليل مقدار الاضطراب مرة بعد مرة أثناء تقسيم الخط .

ولن نتعرض هنا للتفصيلات الرياضية لعملية "الفركتلة" حيث أن هذا الموضوع يعتبر موضوعاً مستقلاً وينطوي على الكثير من المعالجات الرياضية وسوف نكتفى بتقديم الدوال اللازمة للعملية وشرح طريقة استخدامها .

ومن الملاحظ عند استدعاء الدالة Limits مع خط الجبل أن مقياس الاضطراب ومعامل الاضمحلال قد اختلفا عنهما مع خط الساحل . فبالنسبة للجبل استخدمنا القيم :

0.3, 50

والقيمة 0.3 تؤدي إلى تحقيق الاضطراب بطريقة حادة . وكلما زاد هذا العدد أدى إلى اضطراب أكثر سلاسة . لذلك نرى أنه في حالة خط الساحل كانت البارامترات :

0.9, 30

فالقيمة 0.9 تؤدي إلى اضطرابات أقل في الخط . ويمكنك تجربة بعض القيم الأخرى (ما بين الصفر والواحد) وسوف تجد أن القيمة 0 تؤدي إلى أقصى اضطراب ممكن .

[٦] وتقوم الدالة "Limits" بحساب المتغيرات اللازمة للعملية ثم تستخدمها في استدعاء الدالة "Fractalize" . أما دالة الرسم فهي تستخدم دليلاً واحداً وهو الإحداثي y للخط المطلوب رسمه .

أضف إلى قاموسك

Fractal geometry

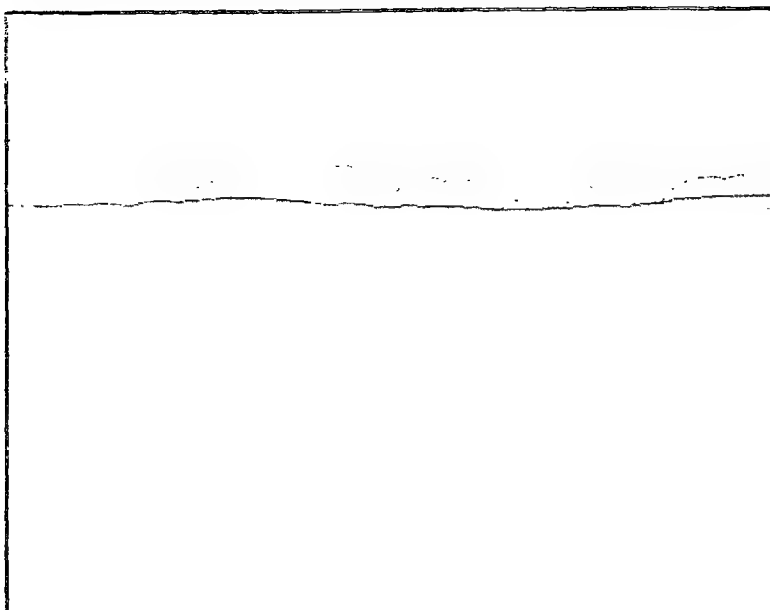
- علم هندسة الفراككتلات

Fractals

- الفراككتلات

Fractalization

- عملية الفركتلة



شكل (٥٠)

ملاحظة

وردت الدالة random في البرنامج الأخير لاستخدامها في توليد الأعداد العشوائية ، ولنا مع هذه الدالة لقاء مفصل في الأبواب القادمة .



الموجز

[١] فى هذا الباب عرفنا البقية الباقية من دوال الرسم للوصلة BGI بلغة نيربوسى++ وقد استخدمناها فى رسم المستطيل، والمضلع، والأقواس، والقطع الناقص، هذا علاوة على دالتى رسم الخط والدائرة التى عرفناها فى الباب الأول :

rectangle	● المستطيل
drawpoly	● المضلع
arc	● القوس
ellipse	● القطع الناقص

[٢] كما عرفنا الكثير من دوال طلاء المساحات الفارغة مثل :

fillpoly	● مضلع مصمت
fillellipse	● قطع ناقص مصمت
sector	● قطاع مصمت
pieslice	● قطاع دائرى مصمت
bar	● قضيب مصمت بدون إطار
bar3d	● قضيب مجسم مصمت

[٣] كما عرفنا كيفية التحكم فى شبكة الطلاء للأشكال المصمتة باستخدام الدالتين :

setfillstyle	● الشبكات الجاهزة
setfillpattern	● الشبكة المبتكرة

وعرفنا أيضاً دالة الطلاء floodfill التى تصلح لطلاء المساحات غير المعروفة مسبقاً والمحدودة بإطار ذى لون معين .

[٤] وفي معرض حديثنا عن دوال الرسم التقينا بدوال نافعة في استخراج المعلومات من الرسم او من المعدات المستخدمة مثل :

- لمعرفة اللون ذي اكبر رقم `getmaxcolor`
- لمعرفة نهايات القوس `getarccoords`

كما عرفنا المنشأ المستخدم لتخزين معلومات القوس المرسوم بالدالة `arc` وهو : `arccoordstype` .

[٥] هذه هي عينات الدوال الواردة في الباب الرابع :

```
#include <graphics.h>                المستطيل
void far rectangle(int left, int top,
                  int right, int bottom);

#include <graphics.h>                المضلع
void far drawpoly(int numpoints,
                 int far *polypoints);

#include <graphics.h>                المضلع المصمت
void far fillpoly(int numpoints,
                 int far *polypoints);

#include <graphics.h>                تحديد شبكة الطلاء
void far setfillstyle(int pattern, int color);

#include <graphics.h>                القوس
void far arc(int x, int y, int stangle,
            int endangle, int radius);

#include <graphics.h>                القطع الناقص
void far ellipse(int x, int y,
                int stangle, int endangle,
                int xradius, int yradius);

#include <graphics.h>                القطع الناقص المصمت
void far fillellipse(int x, int y,
                   int xradius, int yradius);

#include <graphics.h>                القطاع المصمت
void far sector(int x, int y,
               int stangle, int endangle,
               int xradius, int yradius);
```

```

struct arccoordstype {    منشأ إحداثيات القوس
    int x, y;
    int xstart, ystart;
    int xend, yend;
};

#include <graphics.h>    معرفة إحداثيات القوس
void far getarccoords(
    struct arccoordstype far *arccoords);

#include <graphics.h>    معرفة اللون ذي أكبر رقم
int far getmaxcolor(void);

#include <graphics.h>    قطاع دائري
void far pieslice(int x, int y, int stangle,
    int endangle, int radius);

#include <graphics.h>    قضيب مصمت
void far bar(int left, int top,
    int right, int bottom);

#include <graphics.h>    قضيب مجسم مصمت
void far bar3d(int left, int top,
    int right, int bottom,
    int depth, int topflag);

#include <graphics.h>    شبكة الطلاء المبكرة
void far setfillpattern(char far *upattern,
    int color);

#include <graphics.h>    الطلاء بدالة الفيضان
void far floodfill(int x, int y,
    int border);

```



الباب الخامس

أفانين الكتابة في
نسق الرسم



NORMAL NARROW WIDE

SHORT

HIGH

HUGE



مفتتح

إن الرسم لا يقتصر على الدوائر والخطوط والمنحنيات بل يمتد أيضاً ليشمل فن تحسين الخطوط . وفى التراث العربى يعتبر فن تحسين الخطوط العربية من أعرق الفنون التى تميز بها العرب .

وفى مجال دراسة الخطوط فإن للخطوط أشكالاً مختلفة لها أسماء مميزة مثل النسخ والثلث والفارسي والكوفى . وللخطوط اللاتينية أيضاً أشكال وأسماء سوف نلتقى بها فى هذا الباب . وفى مجال الكومبيوتر يطلق على توليفة الشكل مع الحجم اسم "البنط" لأن كل نوع من أنواع الخطوط له حجم سابق التعريف . ومع ذلك ففى هذا الباب أيضاً سوف نلتقى بوسائل تكبير الحروف وكتابتها بالطول والعرض وسائر الأفانين التى تجعل برنامجك فياضاً بالحياة ، وفى الأبواب القادمة سوف نعرف كيف نحرك الحروف على الشاشة !

(٥-١) بنطات الكتابة

كان استخدامنا للدالة cout (أو printf) في نسق الرسم مجرد استخدام مؤقت لما هو متاح من أدوات البرمجة حتى يحين اللقاء مع أدوات معالجة الكتابة الموجودة بالوصلة BGI. وتتضمن دوال الكتابة أدوات مختلفة يمكنك بواسطتها اختيار البنط (Font) وتكبير الحروف ، بل وطباعتها بطريقة رأسية أو أفقية ، كما أن هناك وسائل لابتكار أحجام جديدة للحروف .

وهناك فارق أساسى بين بنط الحروف سابق التعريف والذى يطلق عليه بنط البكسلات (Bitmapped font) ، وبين البنطات التى تمدنا بها الوصلة BGI في نسق الرسم والتى تسمى بنطات الخطوط (stroke fonts) .

فالبنط سابق التعريف يحصر الحرف الواحد في شبكة من البكسلات أبعادها 8×8 وبالتالى فإن المساحة المخصصة لكتابة اللبنة تكون ثابتة دائماً مهما اختلف شكلها .

أما بنطات الخطوط فهى تُرسم خطاً بخط ولذلك فإن المساحة المخصصة للحرف تختلف باختلاف الحرف نفسه . فالحرف I يكتب في مساحة أقل من تلك التى يكتب فيها الحرف M أو الحرف S . علاوة على ذلك فإن تكبير بنطات الخطوط لا يؤدي إلى فقدان أى درجة من الدقة .

والشكل التالى يوضح بعض نماذج الخطوط المكبرة التى يمكن أن نحصل عليها في نسق الرسم .





شكل (١)

(٥-٢) دوال الكتابة على الشاشة *outtext*
outtextxy

تستخدم الدالة *outtext* بطباعة النصوص على الشاشة في الموقع الحالي
لبكسلة الرسم وتأخذ عينة الدالة الصورة الآتية :

```
#include <graphics.h>  
void far outtext(char far *textstring);
```

شكل (٢)

وعندما تستدعى الدالة outtext لطباعة حرفي معين فإن الموقع الحالي لبكسلة الرسم ينتقل إلى ما بعد آخر لبنة تم رسمها على الشاشة .

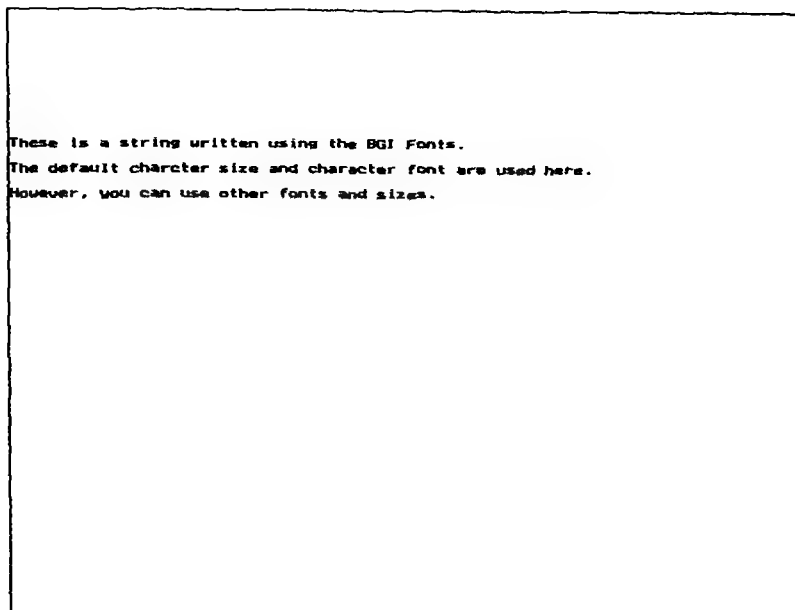
ويجوز تحريك نقطة الرسم باستخدام الدالة moveto قبل استخدام هذه الدالة حتى تبدأ الكتابة عند موقع معين .

ولا نتوقع بطبيعة الحال أنه يمكن استخدام لبنات التحكم التي نستخدمها مع دالة الطباعة printf مثل لبنة الانتقال إلى أول السطر (\n) ، فالدالة outtext هي دالة رسم ولا تتعامل مع لبنات التحكم .

والبرنامج التالي يطبع لنا على الشاشة نصاً مكوناً من ثلاثة أسطر باستخدام الأوضاع سابقة التعريف .

```
/* Program 5-1.cpp */
// Writing text to graphics screen
#include <graphics.h>
#include <conio.h>
main()
{
    int driver, mode;
    driver=DETECT;
    char *string[3];
    int Ystart=100;
    string[0] = "These is a string written\
using the BGI Fonts.";
    string[1] = "The default charcter size\
and character font are used here.";
    string[2] = "However, you can use other\
fonts and sizes.";
    initgraph(&driver, &mode, "d:\\tc\\BGI");
    moveto(0,Ystart);
    for(int i = 0; i <= 2; i++) {
        moveto(0,Ystart+i*20);
        outtext(string[i]);
    }
    getch();
    closegraph();
    return (0);
}
```

شكل (٣)



شكل (٤)

أما الدالة `outtextxy` فهي تستخدم لطباعة حرفي معين في موقع معين على شاشة الرسم . والمقصود بالموقع هو إحداثي البكسل وليس المقصود به الصف والعمود . ولا يؤدي استخدام هذه الدالة إلى تغيير الموقع الحالي للرسم . وهذه هي عينة الدالة .

```
#include <graphics.h>
void far outtextxy(int x, int y,
                  char far *textstring);
```

شكل (٥)

وفي البرنامج التالي نكتب نفس الحرفيات على الشاشة باستخدام الدالة `outtextxy` مع استخدام العداد (i) في ترحيل الكتابة إلى اليمين مع كل سطر .

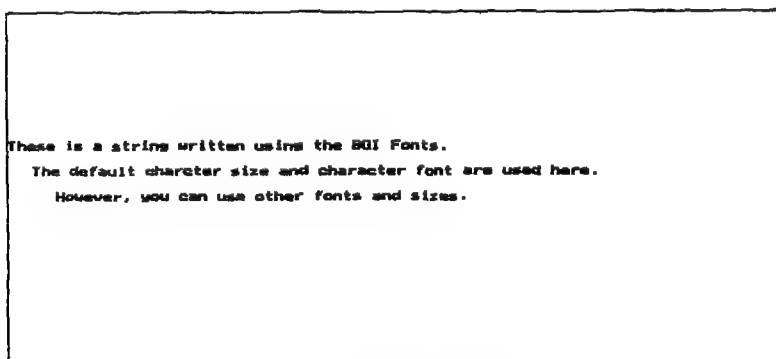
```

/* Program 5-2.cpp */
// Writing text to a pixel location
#include <graphics.h>
#include <conio.h>
main()
{
    int driver, mode;
    driver=DETECT;
    char *string[3];
    int Ystart=100;
    string[0] = "These is a string written\
using the BGI Fonts.";
    string[1] = "The default character size\
and character font are used here.";
    string[2] = "However, you can use other\
fonts and sizes.";
    initgraph(&driver, &mode, "d:\\tc\\BGI");
    for(int i = 0; i <= 2; i++)
        outtextxy(i*20, Ystart+i*20, string[i]);
    getch();
    closegraph();
    return (0);
}

```

شكل (٦)

وهذا هو تنفيذ البرنامج :



شكل (٧)

(٥-٣) التحكم فى البنط والحجم واتجاه الكتابة *settextstyle*

تستخدم الدالة *settextstyle* للتحكم فى شكل اللبئات (المطبوعة بالدوال *outtext* ، *outtextxy*) حيث يمكن بواسطة هذه الدالة اختيار البنط والحجم ، كما يمكن اختيار اتجاه الكتابة (أفقية أو رأسية) . وعينة الدالة *settextstyle* كالآتى :

```
#include <graphics.h>
void far settextstyle(int font,      البنط
                    int direction,  الاتجاه
                    int charsize);  الحجم
```

شكل (٨)

ويمثل البارامتر "font" أحد البنطات المُعرّفة فى الجدول التالى :

DEFAULT_FONT	0	8x8 bit-mapped
TRIPLEX_FONT	1	Stroked triplex
SMALL_FONT	2	Stroked small
SANS_SERIF_FONT	3	Stroked sans-serif
GOTHIC_FONT	4	Stroked gothic

شكل (٩)

ارقام البنطات

والبنط رقم 0 بالجدول (DEFAULT-FONT) هو البنط سابق التعريف وهو من نوع بنط البكسلات أما الأنواع الأربعة الأخرى فهي بنطات الخطوط (stroke) .
أما البارامتر "direction" فيمثل اتجاه الكتابة الذى يأخذ القيم الموضحة بالجدول التالى :

HORIZ_DIR	0	Left to right
VERT_DIR	1	Bottom to top

شكل (١٠)

اتجاه الكتابة

أما بارامتر الحجم (size) فهو يأخذ القيم من صفر إلى 10 ولكل قيمة من القيم تأثير خاص سوف نناقشه فى حينه .

تغيير البنط

فى هذا البرنامج نعرض البنطات المتاحة بالوصلة BGI :

```
/* Program 5-3.cpp */
// Using fonts
#include <graphics.h>
#include <conio.h>
main()
{
    int driver, mode;
    driver = DETECT;
    char *string = "Hello there..";
    initgraph(&driver, &mode, "d:\\tc\\BGI");
    int Xstart = 100;
```

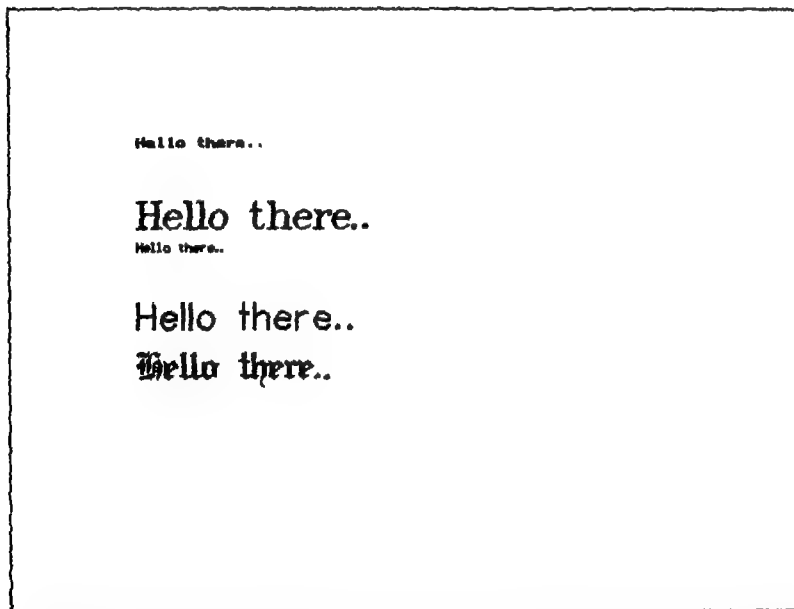
```

int Ystart = 100;
int direction = HORIZ_DIR;
int size = 0;
for(int font = 0; font <= 4; font++) {
    settextstyle(font, direction, size);
    outtextxy(Xstart, Ystart+font*40, string);
}
getch();
closegraph();
return(0);
}

```

شكل (١١)

في هذا البرنامج استخدمنا المتغير "font" (الذي يعبر عن رقم البتط) كعداد يأخذ القيم من 0 إلى 4 . ثم طبعنا العبارة "Hello there.." باستخدام البتطات المتاحة . أما الحجم size فقد منحناه القيمة 0 في الوقت الحالي ، كما استخدمنا الطباعة الأفقية (HORIZ-DIR) . والشكل التالي يوضح المنظر الذي تراه على شاشتك .



شكل (١٢)

فلاش

إن ملفات البنط موجودة في الفهرست الفرعي BGI الموجود مع
تيربو سي++ أو بورلاند سي++ ، وفي حالة عدم تمكن المترجم من
التوصل لملفات البنط فلن تتمكن من استخدام بنطات الخطوط
(stroke) . وهذه الملفات هي :

Directory of D:\TC\BGI

GOTH	CHR	18063	02-28-91
LITT	CHR	5131	02-28-91
SANS	CHR	13596	02-28-91
TRIP	CHR	16677	02-28-91

شكل (١٣)

تأكد من وجود هذه الملفات بالفهرست BGI وتأكد من وجود اسم
الفهرست في العبارة "initgraph" (أو ضع الملفات في الفهرست
الحالي) .

تكبير الحروف

يمكنك تغيير درجة تكبير الحروف بمنح قيم مختلفة للبارامتر "size"
في الدالة settextstyle علماً بأن القيمة "صفر" مخصصة لبنطات الخطوط
(stroke) فقط . أما القيم من 1 إلى 10 فتؤثر على كل من بنطات البكسلات
وبنطات الخطوط . ورغم أن قيمة البارامتر size تؤدي إلى تكبير الحروف
بنسبة معينة ومع ذلك فإن البنطات المختلفة تظهر بأحجام مختلفة .
وهذه هي درجات التكبير المختلفة وتأثير كل منها :

القيمة 0 تؤثر على بنطات الخطوط فقط وتمنحها درجة التكبير سابقة التعريف (4) ، وهي الدرجة التي نراها في تنفيذ البرنامج السابق .

القيمة 1 تعرض الحروف في شبكة 8×8 .

القيمة 2 تعرض الحروف في شبكة 16×16 .

القيمة 3 تعرض الحروف في شبكة 24×24 .

وتزداد قيمة التكبير مع زيادة قيمة البارامتر حتى تصل إلى عشرة أضعاف القيمة الأصلية لحجم الحرف .

والأشكال التالية توضح أحجام الحروف المختلفة من 0 إلى 5 بالنسبة للبنطات الآتية :

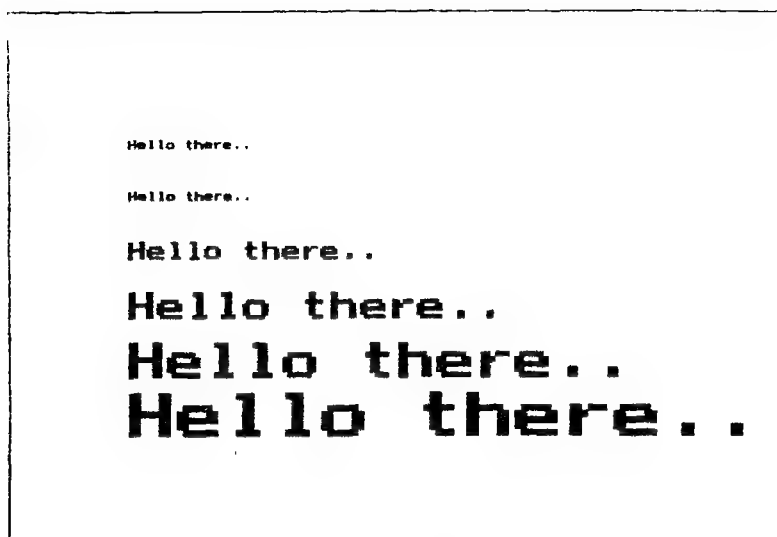
● البنط سابق التعريف (DEFAULT - FONT) انظر شكل (١٤) .

(ونلاحظ في هذا الشكل أنه لا تأثير للقيمة 0 على التكبير) .

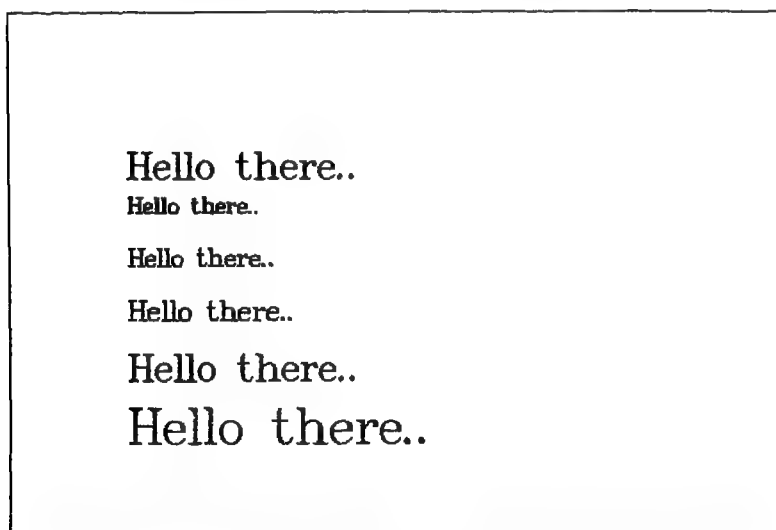
● بنط التربلوكس (TRIPLEX) شكل (١٥) . ونلاحظ تساوى القيمتين 0 ، 4 في درجة التكبير كما ذكرنا .

● البنط القوطي (GOTHIC) شكل (١٦) . وتساوى فيه أيضاً الدرجتان 0 ، 4 كما هو الحال مع سائر بنطات الخطوط .

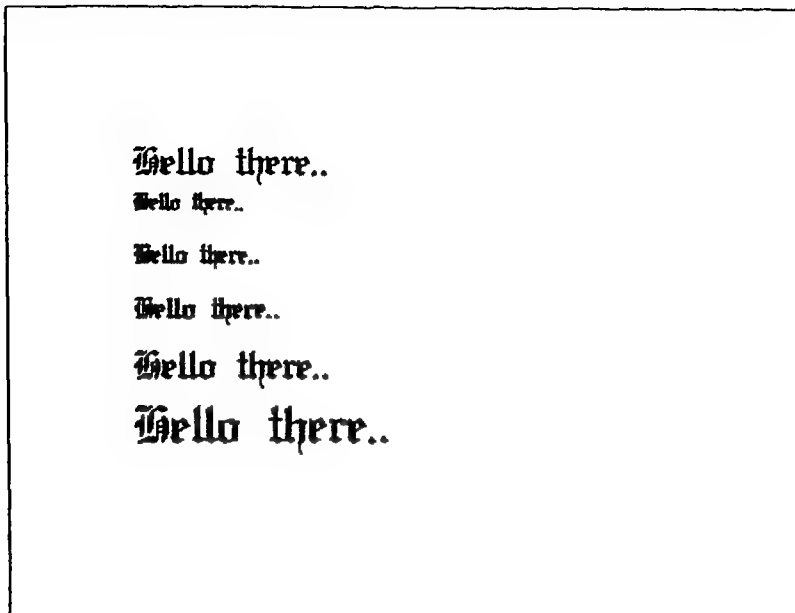




شكل (١٤)
درجات التكبير من 0 إلى 5
وتأثيرها على البنط سابق التعريف



شكل (١٥)
درجات التكبير من 0 إلى 5
وتأثيرها على بنط التريبلكس (TRIPLEX)



شكل (١٦)

درجات التكبير من 0 إلى 5

وتأثيرها على البنط القوطي (GOTHIC)

وبمقارنة الأشكال الثلاثة نجد أن البنط سابق التعريف تقل درجة دقته كلما زادت درجة التكبير ، فهو كما ذكرنا من قبل مكون من عدد ثابت من البكسلات ، بخلاف البنطات الأخرى التي تظهر دائماً بنفس الدقة مع درجات التكبير المختلفة .

وفيما يلي نص البرنامج الذى أنتج الأشكال الثلاثة السابقة فإذا أردت أن تجرب بعض درجات التكبير على البنطات المختلفة عليك بتغيير قيمة المتغير "font" لتغير البنط ، أو قيمتى البداية والنهاية لعدد الحلقة التكرارية "size" لتغير درجة التكبير .

```

/* Program 5-4.cpp */
// Using character amplification
#include <graphics.h>
#include <conio.h>
main()
{
    int driver, mode;
    driver = DETECT;
    char *string = "Hello there..";
    initgraph(&driver, &mode, "d:\\tc\\BGI");
    int Xstart = 100;
    int Ystart = 100;
    int direction = HORIZ_DIR;
    int font =
    for(int size = 0; size <= 5; size++) {
        settextstyle(font, direction, size);
        outtextxy(Xstart, Ystart+size*40, string);
    }
    getch();
    closegraph();
    return(0);
}

```

شكل (١٧)

اتجاه الكتابة

يمكنك اختيار أحد اتجاهين للكتابة بتغيير البارامتر direction كما هو موضح بالجدول شكل (١٠) .

ومن الملاحظ أن الكتابة الرأسية تبدأ دائماً من قاع الشاشة في اتجاه قمته . ولذلك فإذا كان النص المطلوب كتابته رأسياً أطول من ارتفاع الشاشة فإنه سوف يقتطع من بدايته وليس من نهايته .

وفي هذا البرنامج نكتب ثلاث عبارات تبدأ من الإحداثى (100,0) ويفصل بينها وبين بعضها مسافة قدرها 40 بكسلة .

أما البنط المستخدم هنا فهو البنط رقم 3 (SANS-SERIF) وقد تم تحديد

اتجاه الكتابة باستخدام الماكرو :

VERT - DIR

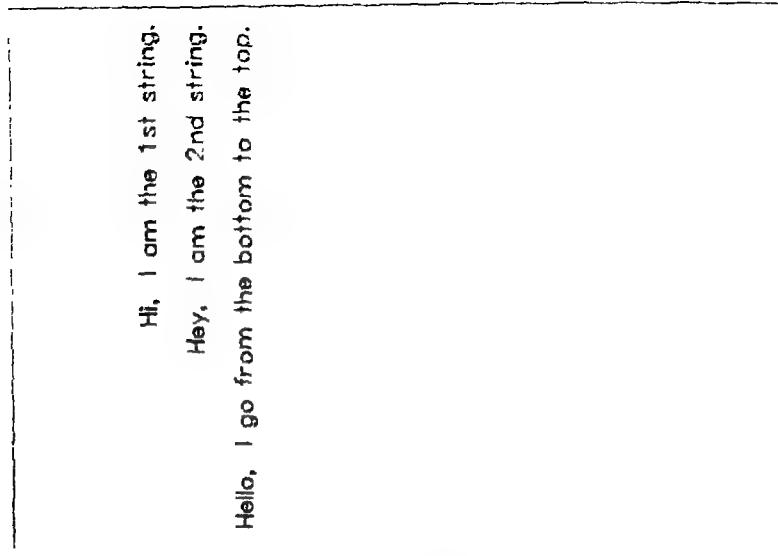
وهذا يكافئ العدد 1 .

```
/* Program 5-5.cpp */
// Vertical text
#include <graphics.h>
#include <conio.h>
main()
{
    int driver, mode;
    driver = DETECT;
    char *string[3];
    string[0] = "Hi, I am the 1st string.";
    string[1] = "Hey, I am the 2nd string.";
    string[2] = "Hello, I go from the bottom\
to the top.";
    initgraph(&driver, &mode, "d:\\tc\\BGI");
    int Xstart = 100;
    int Ystart = 0;
    int font = SANS_SERIF_FONT;
    int direction = VERT_DIR;
    int size = 2;
    settextstyle(font, direction, size);
    for(int i = 0; i <= 2; i++)
        outtextxy(Xstart+i*40, Ystart, string[i]);
    getch();
    closegraph();
    return(0);
}
```

شكل (١٨)

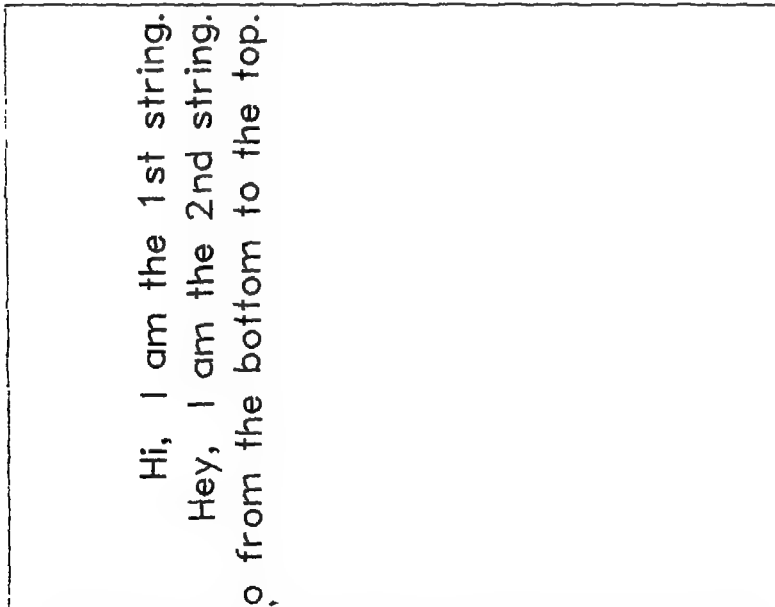
والشكل الآتي يمثل نتيجة التنفيذ :





شكل (١٩)

وبتكبير حجم الخط يمكننا ملاحظة اقتطاع الكتابة من بدايتها كما بالشكل التالي :



شكل (٢٠)

ضبط الهوامش `settextjustify`

إن الوضع سابق التعريف للكتابة الرأسية — كما رأينا — أن تبدأ من أسفل إلى أعلى بحيث يتم تسوية الهوامش عند القمة ، فإذا كان النص طويلاً اقتطع من أسفله (أى من بدايته) .

يمكنك التحكم فى هذا الوضع باستخدام دالة ضبط الهوامش `settextjustify` التى تأخذ الصورة الآتية :

```
#include <graphics.h>
void far settextjustify(int horiz, int vert);
```

شكل (٢١)

وتأخذ الدالة بارامترين :

- `horiz` وهو عدد ثابت يعبر عن طريقة ضبط الهامش فى حالة الكتابة الأفقية .
- `vert` عدد ثابت يعبر عن طريقة الضبط للهامش فى حالة الكتابة الرأسية .

وقيم هذين البارامترين موضحة بالجدول التالى كأعداد وكتابات ماكرو مع توضيح تأثير كل رقم .



horiz *	LEFT_TEXT	0	Left-justify text
	CENTER_TEXT	1	Center text
	RIGHT_TEXT	2	Right-justify text
vert	BOTTOM_TEXT	0	Justify from bottom
	CENTER_TEXT	1	Center text
	* TOP_TEXT	2	Justify from top

شكل (٢٢)

قيم بارامترات تسوية الهوامش

والأوضاع سابقة التعريف لهذه البارامترات هي :

للكتابه الأفقية LEFT-TEXT

للكتابه الرأسية TOP-TEXT

وفي البرنامج التالى نستخدم الدالة `settextjustify` فى ضبط الهامش الأفقى ليكون على اليمين ، والهامش الرأسى ليكون أسفل الشاشة .

وقد تم كتابة العبارة الأفقية "Text Justification" عند الإحداثى الأفقى "getmaxx" وبالتالى فإنها تمتد من اليمين إلى اليسار .

أما العبارات الرأسية فقد استخدمنا نفس العبارات فى المثال السابق ولكن استخدام الهامش `BOTTOM-TEXT` قد أدى إلى تسوية حواف النص عند قاع الشاشة .

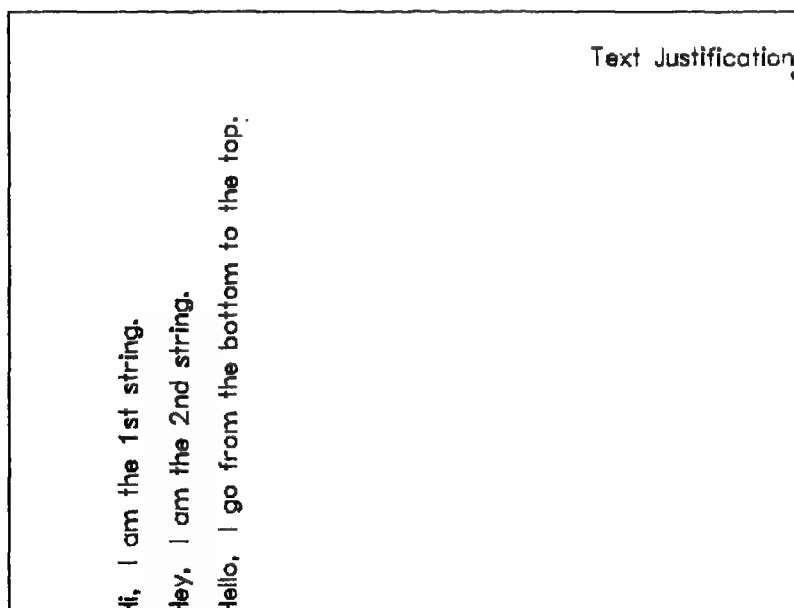
```
/* Program 4-6.cpp */
// Text justification
#include <graphics.h>
#include <conio.h>
main()
{
    int driver, mode;
    driver = DETECT;
    char *string[3];
    string[0] = "Hi, I am the 1st string.";
    string[1] = "Hey, I am the 2nd string.";
```

```

    string[2] = "Hello, I go from the bottom\
to the top.";
    initgraph(&driver, &mode, "d:\\tc\\BGI");
    int Xstart = 100;
    int Ystart = getmaxy();
    int font = SANS_SERIF_FONT;
    int size = 2;
// Justify text:
    settextjustify(RIGHT_TEXT, BOTTOM_TEXT);
// Set font, direction, and size:
    settextstyle(font, HORIZ_DIR, size);
    outtextxy(getmaxx(), 40, "Text Justification");
// Set font, direction, and size:
    settextstyle(font, VERT_DIR, size);
    for(int i = 0; i <= 2; i++)
        outtextxy(Xstart+i*40, Ystart, string[i]);
    getch();
    closegraph();
    return(0);
}

```

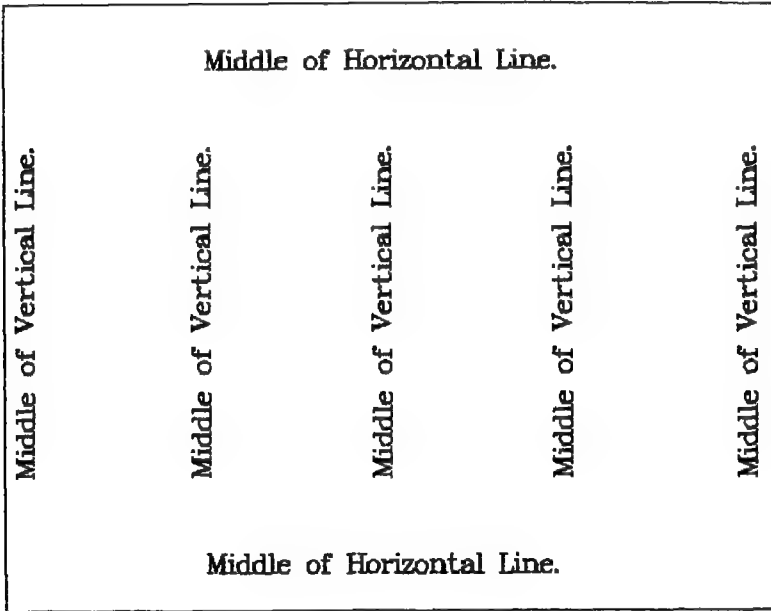
شكل (٢٣)



شكل (٢٤)

تدريب (٥-١)

اكتب البرنامج الذى يؤدي إلى طباعة الحرفيات الموضحة بالشكل
التالى :



شكل (٢٥)

(٥-٤) تخزين واسترجاع أوضاع ضبط الكتابة

`gettextsettings`

إن كل ما يخص ضوابط الكتابة من البنى والاتجاه والتكبير وضبط الهوامش يختزن فى المنشأ `textsettingstype` الذى تم تعريفه فى الملف `graphics.h`.

وتصميم المنشأ كالآتي :

```
#include <graphics.h>
struct textsettingstype {
    int font;
    int direction;
    int charsize;
    int horiz;
    int vert;
};
```

شكل (٢٦)

ويمكن استرجاع البيانات المخزنة في عناصر هذا المنشأ باستخدام الدالة `gettextsettings` التي تأخذ الصورة العامة :

```
#include <graphics.h>
void far gettextsettings(
    struct textsettingstype far *texttypeinfo);
```

شكل (٢٧)

فلو أنك أعلنت في برنامجك عن منشأ من الطراز `textsettingstype` فيمكنك التوصل إلى عناصره باستدعاء الدالة `gettextsettings` .

وفي البرنامج التالي نطبع على شاشة جميع عناصر المنشأ سابقة التعريف (وذلك فور دخول نسق الرسم) ثم نطبعها مرة أخرى بعد تعديلها باستخدام دوال الكتابة والضبط . ويتم الطباعة بالدالة `cout` حيث لا توجد أية رسومات بالبرنامج .

```

/* Program 5-7.cpp */
// Get text settings:
#include <graphics.h>
#include <conio.h>
#include <iostream.h>
void printsettings(void);
main()
{
    int driver, mode;
    driver = DETECT;
    initgraph(&driver, &mode, "d:\\tc\\BGI");
// Print default settings:
    gotoxy(8,6);
    cout << ".Default text settings:\n";
    printsettings();
// Change setting:
    int font = TRIPLEX_FONT;
    int size = 3;
    int direction = VERT_DIR;
    settextjustify(1,1);
    settextstyle(font, direction, size);
// Print new settings:
    gotoxy(8,15);
    cout << ".Text settings now are:\n";
    printsettings();
    getch();
    closegraph();
    return(0);
}
void printsettings(void)
{
    struct textsettingstype P;
    gettextsettings(&P);
    cout << "\tFont = " << P.font << endl;
    cout << "\tDirection = " << P.direction << endl;
    cout << "\tCharacter size = " << P.charsize << endl;
    cout << "\tHorizontal justification = "
        << P.horiz << endl;
    cout << "\tVertical justification = "
        << P.vert << endl;
}

```

شکل (۲۸)

```
.Default text settings:  
Font = 8  
Direction = 8  
Character size = 1  
Horizontal justification = 8  
Vertical justification = 2
```

```
.Text settings now are:  
Font = 1  
Direction = 1  
Character size = 3  
Horizontal justification = 1  
Vertical justification = 1
```

شكل (٢٩)

تدريب (٥-٢)

يمكنك استخدام دوال الطباعة المخصصة للرسم مثل :

`outtextxy`

لتنفيذ البرنامج السابق . وفي هذه الحالة عليك أن تحول الأرقام إلى
حرفيات باستخدام دالة مناسبة مثل `sprintf` أو `itoa` .
جرب وشاهد النتائج .

(٥ - ٥) قياس اتساع وارتفاع الحرفيات

textwidth textheight

عندما نكتب أحد الحرفيات بالبنط العريض فى بيئة الرسم فإنه يحتل مساحة مستطيلة محددة تختلف باختلاف درجة التكبير المستخدمة . وتهتمنا معرفة طول وعرض هذا المستطيل على وجه الخصوص عندما نرغب فى كتابة عدة حروفيات على سطر واحد أو فى عمود واحد (بطريقة رأسية) حتى لا تُكتب الحرفيات فوق بعضها البعض . فنحن عندما نكتب عبارة جديدة نحتاج إلى تحريك نقطة الرسم إلى الموقع الجديد المناسب .

والدالة textwidth ترجع عرض المستطيل الذى يحتله الحرفى ، كما أن الدالة textheight ترجع ارتفاع المستطيل الذى يحتله الحرفى . وعينات هذه الدوال كالآتى :

```
int far textheight(char far *textstring);
int far textwidth(char far *textstring);
```

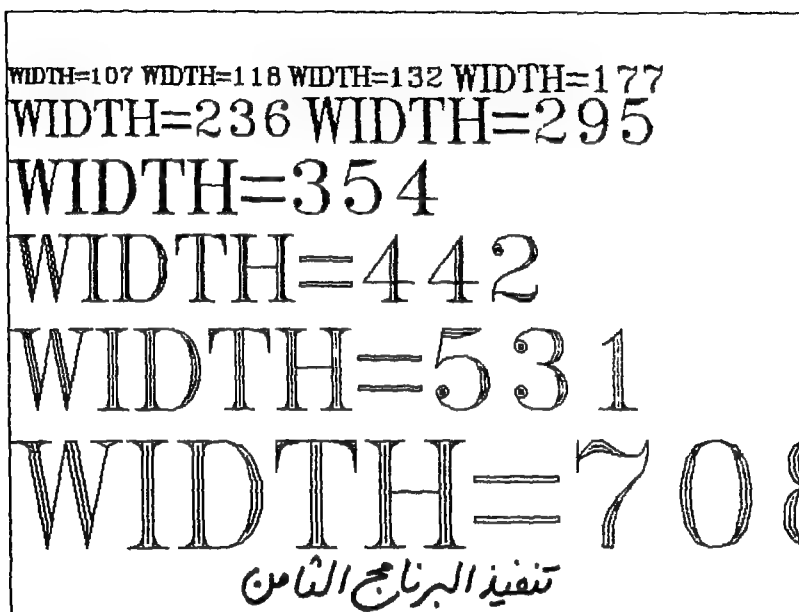
↑ مؤشر إلى الحرف المطلوب قياس أبعاده

شكل (٣٠)

وفى الشكل التالى، فإننا قد كتبنا بعض العبارات المتتابعة على سطر واحد وتحتوى كل عبارة على كلمة "WIDTH=" يعقها الرقم الممثل لطول المستطيل المحتوى على العبارة بما فيها الرقم (الاتساع) .

وقد تمت كتابة الحرفيات المتتابعة بتحريك نقطة الرسم إلى موقع جديد يبعد بمقدار اتساع الحرفى عن الموضع السابق .

أما في حالة العبارات الواقعة على سطور متتالية فهي مفصولة عن بعضها بمقدار ارتفاع آخر حرفي تمت كتابته .



شكل (٣١)

ولو أنك جمعت الأرقام الممثلة للارتفاع في السطر الأول مثلاً :

$$107 + 118 + 132 + 177$$

فإن الناتج يعطى الإحداثى الأفقى الذى يلى الكتابة مباشرة . والشكل التالى يوضح البرنامج المستخدم فى هذا الرسم :




```

/* Program 5-8.cpp */
// String width and height
#include <graphics.h>
#include <stdio.h>
#include <conio.h>
main()
{
    int driver = DETECT, mode;
    int x=0, y;
    char string[80];
    initgraph(&driver, &mode, "d:\\tc\\bgi");
    y = getmaxy()*0.1;
    settextjustify(LEFT_TEXT, CENTER_TEXT);
    for (int size = 1; size <= 10; size++)
    {
        if(size == 5 || size >= 7) {
            x = 0;
            y += textheight(string);
        }
        settextstyle(TRIPLEX_FONT, HORIZ_DIR, size);
        // Make a string using textwidth:
        sprintf(string, "WIDTH=%d",
                textwidth(string));
        // Display the string:
        outtextxy(x, y, string);
        // Move current position to the end of the text:
        x += textwidth(string);
    }
    getch();
    closegraph();
    return(0);
}

```

شكل (٣٢)

وفى إمكانك "رؤية" مواقع نقطة الرسم قبل (أو بعد) كتابة كل عبارة جديدة بإضافة العبارة الآتية :

lineto(0,0);

بداخل الحلقة التكرارية ، فترى بذلك خطأ يشير إلى الموقع الجديد للنقطة .
كما يمكنك طباعة إحداثيات النقط بداخل الحلقة التكرارية أيضاً .

ومن البديهي أنه عند استخدام بنط البكسلات (bitmapped) فإن اتساع الحرف يكون 8 بكسلات وارتفاعه 8 بكسلات أيضاً وذلك عند استخدام معامل التكبير 1 .

نلاحظ أيضاً أن الأرقام في هذا البرنامج قد تم تحويلها إلى حروف لتمهيداً لطباعتها بالدالة `textoutxy` . والدالة المستخدمة في التحويل هي الدالة `sprintf` وهي موجودة بالملف القياسي `stdio.h` وللمرجعة نذكر بعينة هذه الدالة .

```
#include <stdio.h>
int sprintf (char *buffer,
             const char *format [, argument, ...]);
```

شكل (٣٣)

والدالة تأخذ صيغة مماثلة لدالة الطباعة القياسية فيما عدا أنها ترسل الخرج إلى الحرفي المشار إليه بالمؤشر `buffer` . ومن مميزات استخدام هذه الدالة أنها تتمكنك علاوة على تحويل عدد ما إلى حرفي ، من وصل الحرفي الناتج بحرفيات أخرى كما رأينا في البرنامج :

```
sprintf(string, "WIDTH=%d", textwidth(string));
```

(٥ - ٦) الأحجام المبتكرة للخطوط

setusercharsize

تستخدم الدالة `setusercharsize` في التحكم في اتساع وارتفاع الحروف المكتوبة باستخدام الدالة `outtext` (أو `outtextxy`) بحيث يمكنك ابتكار أشكال جديدة للخطوط ، كما أنها تمنحك درجة تكبير (أو تصغير) لأي من الاتساع والارتفاع كلي على حدة . وتؤثر هذه الدالة على بنطات الخطوط فقط وتأخذ عينتها الصورة الآتية :

```
void far setusercharsize(int multx,
                        int divx,
                        {int multy,
                        {int divy});
```

شكل (٣٤)

وتأخذ الدالة أربعة بارامترات تمثل معاملات التكبير والتصغير لكل من الاتساع والارتفاع . ولو افترضنا أننا قد منحنا جميع البارامترات القيمة 1 . فإن الحروف تظهر بالحجم سابق التعريف ، وهو الحجم الذى نحصل عليه لو استخدمنا معامل تكبير قدره "صفر" أو "4" (لاحظ أن معامل التكبير "صفر" يمنح بنطات الخطوط درجة التكبير 4 وهى الدرجة السابقة التعريف) .

ولتكبير الاتساع فإننا نمنح البارامتر الأول multx قيمة عددية أكبر من الواحد تمثل درجة التكبير المطلوبة . فلو استخدمنا الرقم 3 مثلاً أصبح اتساع النص أكبر ثلاثة مرات من درجة الاتساع المعتادة التى نحصل عليها بالدالة . settextstyle

ولو أننا منحنا البارامتر الأول القيمة "1" والثانى القيمة "3" حصلنا على اتساع أصغر ثلاث مرات من الاتساع المعتاد . معنى هذا أن النسبة العددية :

multx

divx

تعبّر عن درجة التكبير . فإذا كانت أقل من الواحد أصبحت تمثل درجة التصغير .

ويسرى نفس المبدأ على النسبة :

multy

divy

وتقبل هذه البارامترات أرقاماً هائلة تصل إلى حدود عرض الشاشة (مثل 639) وارتفاعها مثل (479) بالنسبة للنظام VGAHI . ولا قيمة لهذه الأرقام الكبيرة بطبيعة الحال لأنها أكثر مما يحتاج إليه المبرمج .

ومن البديهي أن هذه الدالة لا تستخدم بمفردها بل يلزم أن تسبقها (مرة واحدة) الدالة `settextstyle` لتحديد البنط ، ودرجة التكبير الأساسية التي تستخدم كقاعدة تعمل من خلالها الدالة `setusercharsize` .

وفي البرنامج التالي نوضح استخدام هذه البارامترات جميعها بطباعة بعض الحرفيات بالصورة المعتادة (كلمة NORMAL) ثم بالصورة المضغوطة الاتساع بنسبة 1:2 (NARROW) ، ثم بالصورة الممتدة الاتساع بمعامل تكبير 4:1 . وعلى السطر التالي يتم ضغط الارتفاع بمقدار الثلث (كلمة SHORT) ثم تكبيره بمقدار 4:1 (HIGH) ثم تكبير العرض والارتفاع معاً بمقدار 4 ، 8 بالترتيب (HUGE) .

وجميع هذه الأرقام منسوبة إلى درجة التكبير المبدئية "4" (الواردة بالبرنامج) .



وفيما يلي نص البرنامج المؤدى إلى هذه الأشكال :

```

/* Program 4-9.cpp */
// User defined character size
#include <graphics.h>
#include <conio.h>
main()
{
    int driver = DETECT, mode;
    initgraph(&driver, &mode, "d:\\tc\\bgi");
    // Text style and initial zise:
    settextstyle(SANS_SERIF_FONT, HORIZ_DIR, 0);
    int x = getmaxx()*0.1;
    int y = getmaxy()*0.1;
    moveto(x,y);
    // Normal text:
    outtext("NORMAL ");
    // Reduce text width by 1/2:
    setusercharsize(1,2,1,1);
    outtext("NARROW ");
    // Magnify width 4 times:
    setusercharsize(4,1,1,1);
    outtext("WIDE ");
    // Reduce height by 1/3:
    moveto(x, y+100);
    setusercharsize(1,1,1,3);
    outtext("SHORT ");
    // Increase height 4 times:
    setusercharsize(1,1,4,1);
    outtext("HIGH ");
    // Increase height 4 times and width 4 times:
    setusercharsize(4,1,8,1);
    outtext(" HUGE");
    getch();
    closegraph();
    return(0);
}

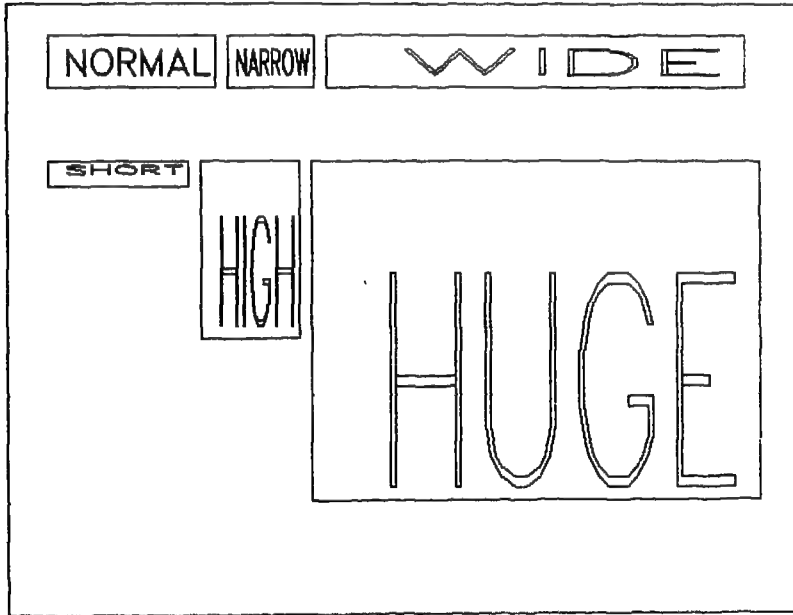
```

شكل (٣٦)

تدريب (٥ - ٣)

على ضوء دراستك في هذا الباب أضف ما يلزم إلى البرنامج التاسع وأجر ما يلزم من تعديلات حتى تصبح كل عبارة مكتوبة محاطة بإطار مستطيل يتناسب مع مساحتها كما في الشكل التالي .

من الضروري أن تلاحظ أن نقطة الرسم تتحرك أفقياً كلما كتبت عبارة في الاتجاه الأفقي (HORIZ - DIR) أما الإحداثي الرأسى فيظل كما هو . استند من دوال إيجاد موقع نقطة الرسم ودوال إيجاد اتساع وارتفاع النص المكتوب .



شكل (٣٧)

الموجز

[١] تعرفنا في هذا الباب بدوال الكتابة ومعالجة النصوص في بيئة الرسم ، فعرفنا كيفية ضبط مواصفات الكتابة لتكون رأسية أو أفقية ؛ كما عرفنا أنواع البنطات الموجودة بالوصلة BGI وكيفية استخدامها . كما عرفنا طرق التكبير والتصغير للحروف .

[٢] فيما يلي ملخص بالدوال التي التقينا بها في هذا الباب .

دالة الكتابة في الموقع الحالي :
`#include <graphics.h>`
`void far outtext(char far *textstring);`

دالة الكتابة في الإحداثي (x,y)
`#include <graphics.h>`
`void far outtextxy(int x, int y,`
`char far *textstring);`

دالة ضبط مواصفات الكتابة
`#include <graphics.h>`
`void far settextstyle(int font,`
`int direction,`
`int charsize);`

دالة ضبط الهوامش
`#include <graphics.h>`
`void far settextjustify(int horiz, int vert);`

دالة التعرف على مواصفات الكتابة الحالية
`#include <graphics.h>`
`void far gettextsettings(`
`struct textsettingstype far *texttypeinfo);`

منشأ مواصفات الكتابة
`#include <graphics.h>`
`struct textsettingstype {`
`int font;`
`int direction;`

```
int charsize;
int horiz;
int vert;
};
```

دالة إيجاد ارتفاع واتساع الحرفي `<graphics.h>`

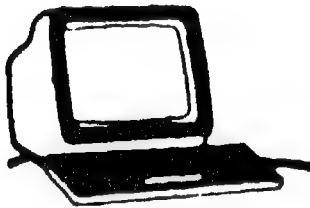
```
int far textheight(char far *textstring);
int far textwidth(char far *textstring);
```

دالة تحويل الأعداد إلى حرفيات وتوصيلها معاً

```
#include <stdio.h>
int sprintf (char *buffer,
             const char *format [, argument, ...]);
```

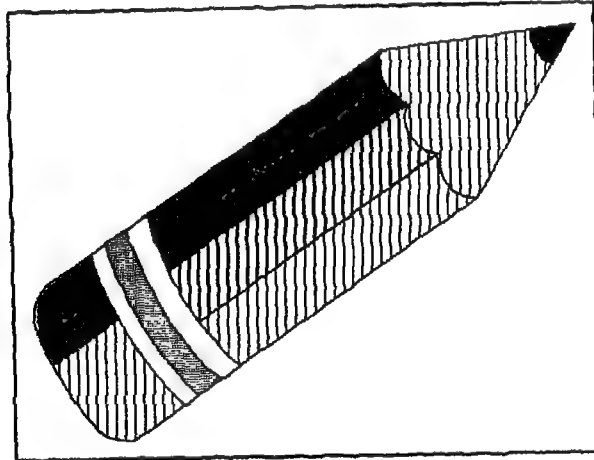
دالة التكبير والتصغير لحجم الكتابة `<stdio.h>`

```
void far setusercharsize(int multx,
                        int divx,
                        int multy,
                        int divy);
```



الباب السادس

مهارات عامة في الرسم



مفتتح

نلتقى فى هذا الباب بموضوعات متفرقة تهدف إلى تعميق المفاهيم الخاصة ببيئة الرسم ، وإلى إتقان البرنامج .

كما سنلتقى فى هذا الباب أيضاً بالنوافذ فى بيئة الرسم (viewports) وكذلك بصفحات الذاكرة التى تمكن المبرمج من إنشاء أكثر من شاشة واحدة فى نفس الوقت يستطيع التنقل بينها . وهذه المهارة الأخيرة هى إحدى طرق برمجة الأشكال المتحركة التى سنلتقى بها لقاءً شاملاً فى الباب القادم .

(٦ - ١) روتينات معالجة الأخطاء graphresult

grapherrormsg

قد يسهو على المبرمج في بعض الأحيان فلا يكتب اسم الفهرست المحتوى على ملفات قيادة أجهزة الفيديو أو يكتب الاسم بطريقة غير صحيحة . وفي هذه الحالة فإن المترجم يرسل رسالة بالخطأ لكنها تمر مروراً سريعاً على الشاشة لا يستطيع معه المبرمج أن يتحقق مما يحدث ثم تعود دفعة التحكم إلى محرر بورلاند سي++ (أو تيربو سي++) .

ومن المفضل أن يحتوى أى برنامج للرسم على روتين خاص لاكتشاف مثل هذا الخطأ وإرسال الرسالة المناسبة مع منح المبرمج الفرصة الكافية لقراءة الرسالة والتحقق فيها ؛ فإذا انتهى منها ضغط على أى زر لكي يعود إلى بيئة المحرر .

وشريحة البرنامج الآتية يمكن استخدامها في هذا الغرض حيث نبدأ بها برنامجنا في مجال الرسم دائماً .

```
/* Program 6-1.cpp */
// Error handling
#include <graphics.h>
#include <conio.h>
#include <stdlib.h>
#include <iostream.h>
main()
{
    int driver = DETECT;
    int mode;
    int errorcode;
    initgraph(&driver, &mode, "z:\\\\tc\\bgi");
    // Read result of initialization
    errorcode = graphresult();
    if (errorcode != grOk) {
        cout << "Graphics error:"
             << grapherrormsg(errorcode) << endl
             << "Make sure the path to graphics "
```

```

        << "drivers is correct." << endl
        << "Press any key to exit";
    getch();
    clrscr();
    exit(1);
}
return(0);
}

```

شكل (١)

وكما نرى فى البرنامج فإن هناك دالة مخصصة للتعرف على نوع الخطأ الحادث هى الدالة "graphresult" وتأخذ الصورة التالية :

```

#include <graphics.h>
int far graphresult(void);

```

شكل (٢)

وترجع هذه الدالة رقماً كودياً يمثل الخطأ الحادث (سيلي عرض أكواد الأخطاء) . فإذا أردت شرحاً لنوع الخطأ الحادث بدون استخدام جدول الأرقام الكودية للأخطاء فعليك أن تستخدم القيمة المرتجعة من الدالة السابقة كبارامتر للدالة "grapherrormsg" فهذه الدالة ترجع مؤشراً إلى الحرفي الممثل لنوع الخطأ .

وتأخذ عينة الدالة الأخيرة الصورة :

```

#include <graphics.h>
char *far grapherrormsg(int errorcode);

```

↑ الرقم الكودى للخطأ

شكل (٣)

والجدول التالى يوضح أرقام الأخطاء والحرفي الذى يشرح كلاً منها :

0	grOk	No error
-1	grNoInitGraph	(BGI) graphics not installed (use initgraph)
-2	grNotDetected	Graphics hardware not detected
-3	grFileNotFound	Device driver file not found
-4	grInvalidDriver	Invalid device driver file
-5	grNoLoadMem	Not enough memory to load driver
-6	grNoScanMem	Out of memory in scan fill
-7	grNoFloodMem	Out of memory in flood fill
-8	grFontNotFound	Font file not found
-9	grNoFontMem	Not enough memory to load font
-10	grInvalidMode	Invalid graphics mode for selected driver
-11	grError	Graphics error
-12	grIOerror	Graphics I/O error
-13	grInvalidFont	Invalid font file
-14	grInvalidFontNum	Invalid font number
-15	grInvalidDeviceNum	Invalid device number
-18	grInvalidVersion	Invalid version number

شكل (٤)

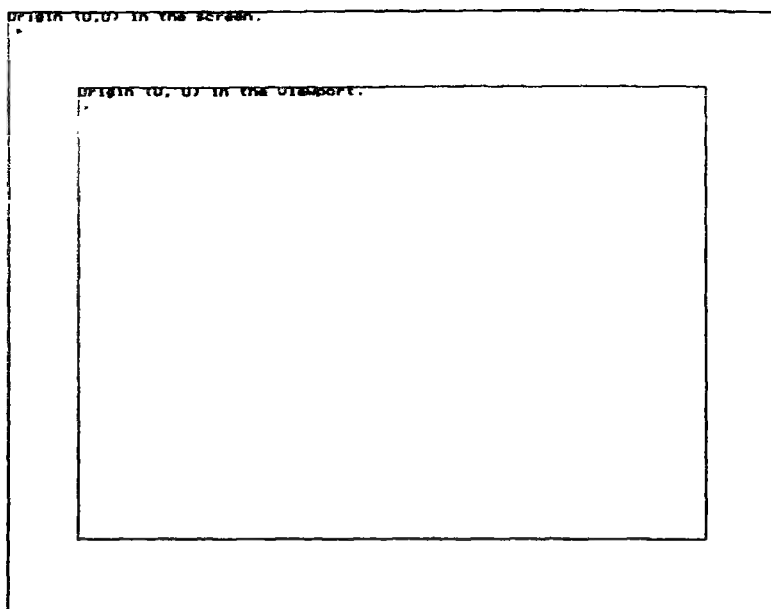
ارقام الأخطاء ومعناها

وكما نرى فى الجدول فإن لكل رقم من أرقام الأخطاء "ماكرو" مكافئ له وهو يساعد على تذكر نوع الخطأ حيث أن اسم الماكرو معبر عن معناه بصورة مختصرة . فالماكرو "grOK" (وهو الماكرو المستخدم فى البرنامج) هو مختصر العبارة : "graphics OK" بمعنى أن إجراءات الدخول فى نسق الرسم على ما يرام . وقد استخدمنا فى البرنامج اسم هذا الماكرو مسبقاً بمؤثر النفى المنطقى (NOT) لأن العبارة الشرطية تتحقق فى حالة وجود خطأ .

(٦ - ٢) النوافذ فى نسق الرسم *setviewport*

يمكنك - كما فى نسق الكتابة - إنشاء النوافذ فى نسق الرسم وذلك باستخدام الدالة *setviewport* . ويطلق على النافذة فى نسق الرسم الاسم "View Port" تمييزاً لها عن نافذة النصوص .

ومن الجدير بالذكر أنه عند عدم إنشاء أية نوافذ فى البرنامج فإن الشاشة كلها تصبح هى النافذة (سابقة التعريف) . وفى الشكل التالى بدأنا برسم مستطيل فى النافذة سابقة التعريف من النقطة (0,0) إلى أقصى حدود الشاشة ، ثم كتبنا العبارة الموضحة عند نقطة الأصل (0,0) . أعقب ذلك إنشاء النافذة الصغيرة التى تبعد بمقدار 60 بكسلة عن حواف الشاشة من كل جانب . نتيجة لذلك فإن نقطة الأصل قد انتقلت هى الأخرى وأصبحت هى النقطة (60,60) .



شكل (٥)

وبمعنى آخر فإن أى إشارة إلى نقطة الأصل (0,0) بعد فتح النافذة تعنى الركن الأيسر العلوى للنافذة الحالية .

وقد كتبنا عبارة ثانية عند نقطة الأصل الجديدة فظهرت في الركن الأيسر العلوى . أما دوال الرسم جميعاً فإنها سوف تعمل من خلال النافذة الحالية المفتوحة كما لو كانت هى شاشة الرسم .

والدالة المستخدمة في فتح النوافذ تأخذ الصورة الآتية :

```
#include <graphics.h>      // ركن الأيسر العلوى سم
void far setviewport(int left, int top,
                    int right, int bottom,
                    int clip);
```

شكل (٦)

حيث :

left, top إحداثى الركن الأيسر العلوى للنافذة .

right, bottom إحداثي الركن الأيمن السفلي للنافذة .

أما البارامتر clip فهو يأخذ أحد قيمتين صفر أو 1 .

وهو يسمى بارامتر القص . فإذا كانت قيمة بارامتر القص صفراً فإن محتويات النافذة من الرسومات تمتد إلى خارج حدودها . أما إذا كانت قيمة بارامتر القص 1 فإن الرسومات الموجودة بالنافذة يتم قطعها عند حدود النافذة بحيث لا تتعدى مساحتها .

والشكل التالي يوضح البرنامج الذي رسمنا به النافذة الموضحة بالشكل السابق على بعد 60 بكسلة من حواف الشاشة .

```
/* Program 6-2.cpp */
// Viewports
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <iostream.h>
main()
{
    int driver = DETECT;
    int mode;
    int errorcode;
    initgraph(&driver, &mode, "D:\\tc\\bgi");
    // Read result of initialization
    errorcode = graphresult();
    if (errorcode != grOk) {
        cout << "Graphics error:"
             << grapherrormsg(errorcode) << endl
             << "Make sure the path to graphics "
             << "drivers is correct." << endl
             << "Press any key to exit";
        getch();
        clrscr();
        exit(1);
    }
}
```



```

    }
    setcolor(YELLOW);
    outtextxy(0,0,"Origin (0,0) in the screen.");
    // Draw a rectangle:
    setcolor(WHITE);
    rectangle(0,0, getmaxx(), getmaxy());
    // Create a viewport:
    setviewport(60, 60, getmaxx()-60,
               getmaxy()-60, 1);
    setcolor(YELLOW);
    outtextxy(0,0,"Origin (0, 0) in the viewport.");
    // Draw a rectangle in the new viewport:
    setcolor(WHITE);
    rectangle(0,0, getmaxx()-120, getmaxy()-120);
    getch();
    closegraph();
    return(0);
}

```

شكل (٧)

ولا يفوتنا ملاحظة أن هناك عبارتين ، تمت كتابة كل منهما بدءاً من نقطة الأصل (0,0) باستخدام الدالة outtextxy ، ومع ذلك فإن الأولى ظهرت عند ركن الشاشة ، والثانية قد ظهرت عند ركن النافذة .

كما نلاحظ أن المربع الأول قد تم رسمه باستخدام الإحداثيات :

● (0,0) للركن الأيسر العلوى .

● (getmaxx(), getmaxy()) للركن الأيمن السفلى .

أما المستطيل الثانى فقد تم رسمه بالإحداثيات :

● (0,0) للركن الأيسر العلوى (نقطة الأصل الجديدة) .

● (getmaxx() - 120, getmaxy() - 120)

والسبب فى طرح العدد 120 هو أن قيمة كل من getmaxx ، getmaxy

لم تتغير بعد إنشاء النافذة .

viewporttype
getviewsettings

مُنشأ معلومات النافذة

تخزن جميع المعلومات الخاصة بالنافذة الحالية في المنشأ
viewporttype كالآتي :

```
#include <graphics.h>
struct viewporttype {
    int left;
    int top;
    int right;
    int bottom;
    int clip;
};
```

شكل (٨)

ويمكنك التوصل إلى هذه المعلومات باستخدام الدالة getviewsettings
التي تأخذ الصورة العامة :

```
#include <graphics.h>
void far getviewsettings (
    struct viewporttype far *viewport);
```

شكل (٩)

وكما نرى في عينة الدالة أنها تستخدم مؤشراً يشير إلى المنشأ المحتوى
على معلومات النافذة .

فلو أنك أعلنت عن منشأ P (مثلاً) من الطراز viewporttype فتصبح
معلومات النافذة كالآتي :

- P.left الإحداثى الأفقى للركن الأيسر العلوى .
- P.top الإحداثى الرأسى للركن الأيسر العلوى .
- P.right الإحداثى الأفقى للركن الأيمن السفلى .
- P.bottom الإحداثى الرأسى للركن الأيمن السفلى .
- P.clip قيمة بارامتر القص .

ومن المهم أن تلاحظ أن معلومات النافذة غير منسوبة للنافذة بمعنى أن إحداثيات نقطة الأصل (P.left, P.top) لن تكون (0,0) بل ستكون هي الإحداثيات المطلقة (المنسوبة للشاشة) .

كما أن بارامتر القص يقبل أى قيمة غير صفرية فلا يشترط بالضرورة أن تكون القيمة غير الصفرية هي 1 .

ويمكنك إضافة بعض عبارات الطباعة إلى البرنامج السابق للتحقق من هذه المعلومات .

مسح محتويات النافذة	<code>clearviewport</code>
أو محتويات الشاشة	<code>cleardevice</code>

يمكنك أن تمسح الرسومات الموجودة بالنافذة المفتوحة حالياً دون المساس بما هو خارج النافذة وذلك باستخدام الدالة `clearviewport` التى تأخذ الصورة :

```
#include <graphics.h>
void far clearviewport(void);
```

شكل (١٠)

أما الدالة `cleardevice` فهى تؤدي إلى مسح الشاشة كلها بما فيها من نوافذ وتنقل نقطة الرسم إلى الموقع الأصلى (0,0) وعينة هذه الدالة كالآتى :

```
#include <graphics.h>
void far cleardevice(void);
```

شكل (١١)

وحتى نستطيع الإلمام بخصائص النوافذ فلنضع النقط على الحروف في هذا المثال الشامل .

عند تنفيذ هذا البرنامج سوف تحصل على الشكل التالى الذى يحتوى على الآتى :

- مستطيل يقع على حدود النافذة سابقة التعريف (كما سبق) ، علاوة على عبارة مكتوبة عند نقطة الأصل .

- مجموعة من الدوائر المتمركزة عند منتصف الشاشة ، وتمتد فى أقطارها حتى حدود الشاشة الرأسية .

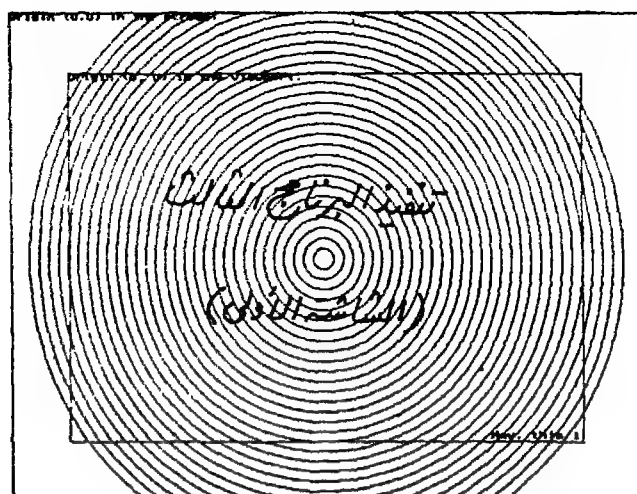
- مستطيل يقع على حدود النافذة الجديدة التى بموجبها انتقلت نقطة الأصل إلى الإحداثى (60,60) كما سبق ، مع كتابة عبارة بداخلها عند نقطة الأصل .

- نرى بداخل النافذة عبارة مقطوعة الذيل :

Hey, this is the end of the viewport

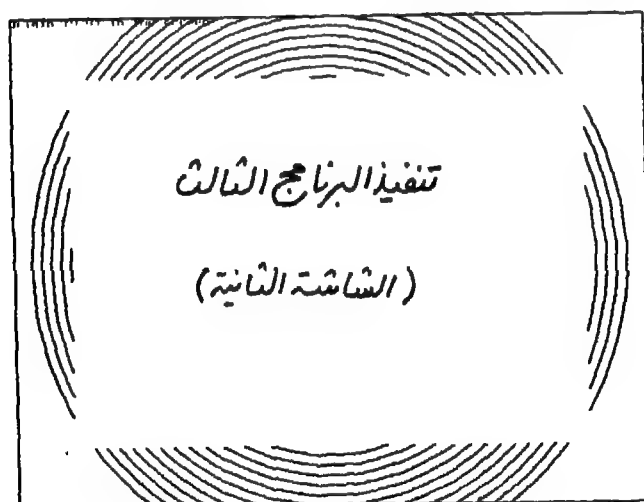
فقد ظهر من العبارة الجزء المكتوب بالبنط الأسود فقط أما الباقي فتم قطعه لوقوعه خارج النافذة .

وعند الضغط على أى زر سوف تشاهد محتويات النافذة تُمسح تماماً ، سواء كانت المحتويات تابعة للنافذة مثل العبارة المقطوعة ، أو كانت المحتويات تابعة للشاشة الكبيرة مثل الدوائر . وهذا هو الشكل الذى تحصل عليه عند الضغط على أى زر :



شكل (١٢)

والشكل التالي يوضح البرنامج الذى أنتاج هذه الشاشات :



شكل (١٣)

```
/* Program 6-3.cpp */
// Viewports
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
```

```

#include <conio.h>
#include <iostream.h>
main()
{
    int driver = DETECT;
    int mode;
    int errorcode, radius;
    viewporttype P;
    initgraph(&driver, &mode, "D:\\tc\\bgi");
    // Read result of initialization:
    errorcode = graphresult();
    if (errorcode != grOk) {
        cout << "Graphics error:"
            << grapherrormsg(errorcode) << endl
            << "Make sure the path to graphics "
            << "drivers is correct." << endl
            << "Press any key to exit";
        getch();
        clrscr();
        exit(1);
    }
    setcolor(YELLOW);
    outtextxy(0,0,"Origin (0,0) in the screen.");
    setcolor(WHITE);
    for(radius=10; radius <= 300; radius += 10)
        circle(getmaxx()/2,getmaxy()/2,radius);
    rectangle(0,0, getmaxx(), getmaxy());
    // Create a viewport:
    setviewport(60, 60, getmaxx()-60,
               getmaxy()-60, 1);
    setcolor(YELLOW);
    outtextxy(0,0,"Origin (0, 0) in the viewport.");
    // Draw a rectangle in the new viewport:
    setcolor(WHITE);
    rectangle(0,0, getmaxx()-120, getmaxy()-120);
    // Get viewport info:
    getviewsettings(&P);
    moveto(P.right-150, P.bottom-70);
    setcolor(YELLOW);
    // Display text inside the window:
    outtext("Hey, this is the end of the viewport");
    getch();
    clearviewport();
    getch();
    closegraph();
    return(0);
}

```

شكل (١٤)

مناقشة البرنامج : (ملاحظة : استعن بالأرقام فى تتبع البرنامج)

[١] بدأ البرنامج بالإعلان عن المتغيرات شاملة المنشأ "P" من "النمط" "viewporttype" ، وهو يحتوى على معلومات النافذة المفتوحة . ذلك روتين الأخطاء كالمعتاد .

[٢] ثم بعد ذلك رسم الدوائر والمستطيل وكتابة العبارة الأولى التى تظهر أعلى الشاشة .

[٣] تم إنشاء النافذة على بعد 60 بكسلة من حواف الشاشة ، كما فى البرنامج السابق ، ثم رسمنا على حدودها المستطيل الصغير .

[٤] تم استدعاء الدالة getviewsettings لقراءة معلومات النافذة المخزنة فى المنشأ .

وبذلك أصبحت المتغيرات left ، top ، bottom ، right كلها معلومة للبرنامج . ولو أنك طبعت إحداثيات الركن الأيمن السفلى باستخدام المتغيرات bottom ، right فسوف تجد أنها (579,419) وذلك بالنسبة لكارت الرسم . VGA

[٥] تم بعد ذلك الانتقال إلى نقطة قريبة من قاع النافذة باستخدام الدالة moveto تمهيداً لكتابة العبارة الموضحة ، والتى سوف تظهر مقطوعة بسبب استخدام بارامتر القص 1 (عند إنشاء النافذة) .

[٦] عند الضغط على أى زر استجابة للدالة getch فإن الدالة clearviewport تؤدي إلى مسح محتويات النافذة .

إعادة النوافذ إلى الوضع سابق التعريف graphdefaults

بعد إنشاء نافذة أو أكثر ، فلعلك ترغب فى العودة إلى النافذة الأصلية (الشاشة) للعمل فيها من جديد . إن الدالة graphdefaults تستخدم فى هذا الغرض وهى تؤدي إلى النتائج الآتية :

- تجعل النافذة الحالية هي الشاشة كلها .
- تنقل الموقع الحالي لنقطة الرسم إلى الموقع (0,0) .
- تعيد الألوان إلى الوضع سابق التعريف .
- تعيد شبكة الطلاء إلى الوضع سابق التعريف .
- تعيد خصائص الكتابة (البنط والهوامش والحجم) إلى الوضع سابق التعريف .

وتأخذ عينة الدالة الصورة الآتية :

```
#include <graphics.h>
void far graphdefaults(void);
```

شكل (١٥)

ولو أنك ، في البرنامج السابق ، استبدلت الدالة `clearviewport` بالدالة `cleardevice` فسوف يؤدي ذلك إلى مسح الشاشة كلها عند الضغط على أى زر .

أما لو استبدلتها بالدالة `graphdefaults` فلن يتغير المشهد على الشاشة ولكن أى كتابة جديدة (أو رسم) سوف تظهر منسوبة للشاشة الكبيرة وبالخصائص سابقة التعريف .

ومعنى ذلك أيضاً أن استخدام دالة مسح النافذة `clearviewport` تاليةً للدالة `graphdefaults` سوف يؤدي إلى مسح الشاشة كلها لأن النافذة الوحيدة في هذه الحالة هي الشاشة . والبرنامج التالى هو تطوير للبرنامج السابق ولكنه يهدف إلى التعرف بخصائص الدوال :

graphdefaults

clearviewport

cleardevice

وقد قسمنا منطق البرنامج إلى جزئين : دالة رئيسية ، ودالة لرسم النوافذ وكتابة النصوص بداخلها . وهذا هو منطق الدالة الرئيسية :

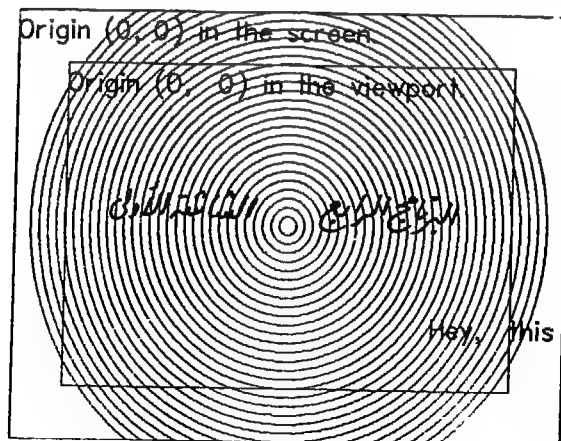

```
// Set up the text font and size:
settextstyle(SANS_SERIF_FONT, HORIZ_DIR, 0);
MakeWindow();
// Erase the screen:
cleardevice();
// Reset the fonts:
graphdefaults();
MakeWindow();
// Reset the viewport:
graphdefaults();
clearviewport();
outtextxy(30,30,"Back to defaults. Now,\
I am in the big screen");
getch();
closegraph();
return(0);
```

شكل (١٦)

وفيما يلي نعلق على الخطوات المشار إليها بالأرقام فى شريحة البرنامج :

[١] يبدأ البرنامج بتحديد البنط والاتجاه وحجم الخط .

[٢] يلي ذلك استدعاء الدالة "MakeWindow" وهى ترسم النافذة التى رسمناها من قبل فيما عدا أننا قد حددنا بارامتر القص بالرقم "صفر" . وبالتالى فإن محتويات النافذة تمتد إلى خارجها كما بالشكل التالى :

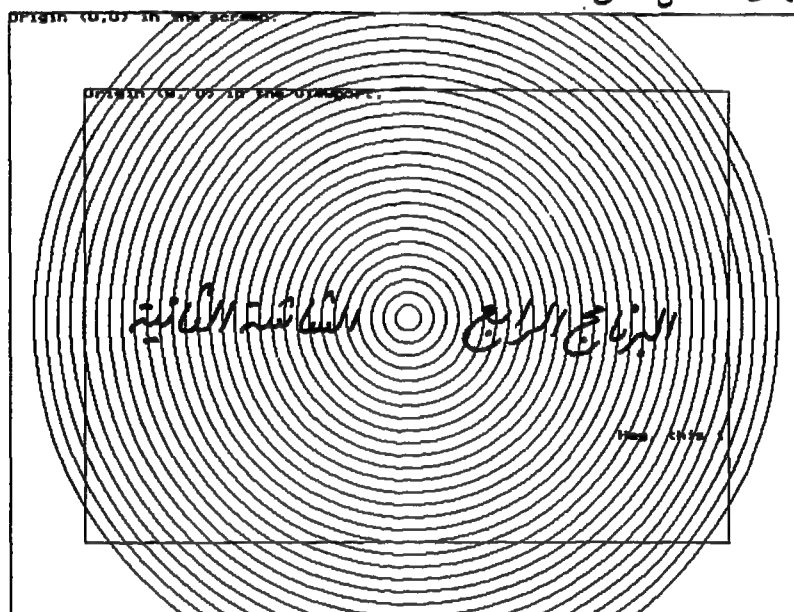


شكل (١٧)

[٣] عند الضغط على أى زر ننتقل إلى الخطوة الثالثة حيث يتم مسح الشاشة بالدالة `cleardevice` .

[٤] فى الخطوة الرابعة نعيد جميع المواصفات سابقة التعريف باستخدام الدالة `graphdefaults` .

[٥] تستدعى دالة رسم النافذة مرة أخرى ، ولكنها فى هذه الحالة سوف تمنحنا مشهداً مختلفاً حيث تظهر الخطوط بالحجم الصغير والبنط سابق التعريف كما فى الشكل التالى :



شكل (١٨)

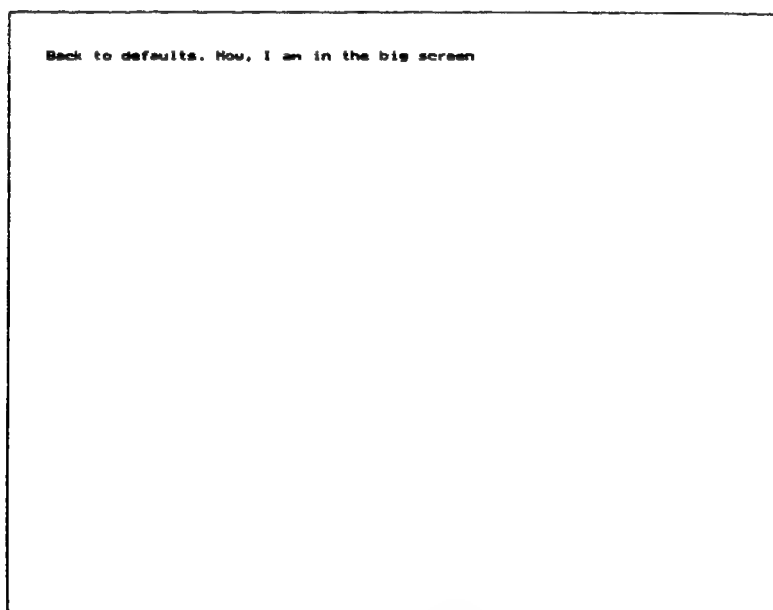
ونلاحظ هنا أنه لو لم يتم مسح الشاشة فى الخطوة رقم (٣) فإن المشهد الجديد للنافذة بمحتوياتها سوف يظهر فوق المشهد القديم .

(جرب حذف الدالة `cleardevice`) .

[٦] عند الضغط على أى زر تنتهى الدالة ثم نعيد المواصفات سابقة التعريف مرة أخرى بالدالة `graphdefaults` وبالتالى تصبح لدينا نافذة واحدة هى الشاشة .

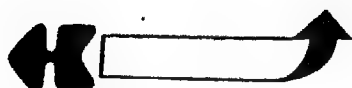
[٧] عند مسح النافذة الحالية باستخدام الدالة clearviewport فإن هذا يؤدي إلى مسح الشاشة كلها .

[٨] وأخيراً نطبع النص الموضح بالبرنامج على شاشة نظيفة فيظهر باللون سابق التعريف ، وبالحجم الصغير ، عند الإحداثي (30,30) كما بالشكل التالي :



شكل (١٩)

وفيما يلي نقدم نص البرنامج كاملاً :



```

/* Program 6-4.cpp */
// Reset viewports
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <iostream.h>
void MakeWindow(void);
main()
{
    int driver = DETECT;
    int mode;
    int errorcode;
    initgraph(&driver, &mode, "D:\\tc\\bgi");
// Read result of initialization:
    errorcode = graphresult();
    if (errorcode != grOk) {
        cout << "Graphics error:"
             << grapherrormsg(errorcode) << endl
             << "Make sure the path to graphics "
             << "drivers is correct." << endl
             << "Press any key to exit";
        getch();
        clrscr();
        exit(1);
    }
// Set up the text font and size:
    settextstyle(SANS_SERIF_FONT, HORIZ_DIR, 0);
    MakeWindow();
// Erase the screen:
    cleardevice();
// Reset the fonts:
    graphdefaults();
    MakeWindow();
// Reset the viewport:
    graphdefaults();
    clearviewport();
    outtextxy(30,30,"Back to defaults. Now,\
I am in the big screen");
    getch();
    closegraph();
    return(0);
}

```

شکل (۲۰)

```
// The viewport function:
void MakeWindow(void)
{
    int radius;
    viewporttype P;
    setcolor(YELLOW);
    outtextxy(0,0,"Origin (0,0) in the screen.");
    setcolor(WHITE);
    for(radius=10; radius <= 300; radius += 10)
        circle(getmaxx()/2,getmaxy()/2,radius);
    rectangle(0,0, getmaxx(), getmaxy());
    // Create a viewport:
    setviewport(60, 60, getmaxx()-60,
               getmaxy()-60, 0);
    setcolor(YELLOW);
    // cleardevice();
    outtextxy(0,0,"Origin (0, 0) in the viewport.");
    // Draw a rectangle in the new viewport:
    setcolor(WHITE);
    rectangle(0,0, getmaxx()-120,
               getmaxy()-120);
    // Get viewport info:
    getviewsettings(&P);
    moveto(P.right-150, P.bottom-150);
    setcolor(YELLOW);
    // Display text inside the window:
    outtext("Hey, this is the end of the viewport");
    getch();
}
```

شكل (٢١)

(٦ - ٣) استخدام صفحات الذاكرة
setactivepage
setvisualpage

يمكنك أن ترسم في أكثر من صفحة وتحفظ بالرسومات جميعاً في الذاكرة في نفس الوقت كما يمكنك أن تنتقل بين الصفحات المختلفة فتعرض محتوياتها على الشاشة .

والمترجم تيريو سي++ يستخدم الصفحة رقم "0" كصفحة سابقة التعريف .

وتستخدم الدالة `setactivepage` لاختيار الصفحة العاملة بحيث أن جميع أوامر الرسم التالية تتجه إلى هذه الصفحة . كما تستخدم الدالة `setvisualpage` لعرض محتويات الصفحة على الشاشة .
وهذه هي عينات الدوال :

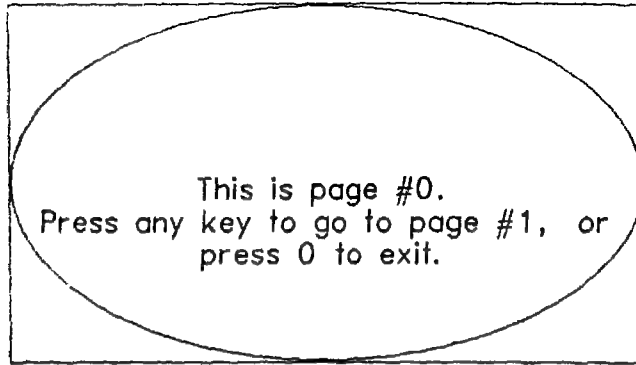
```
#include <graphics.h>
void far setactivepage(int page);
void far setvisualpage(int page);
```

شكل (٢٢)

فلاش

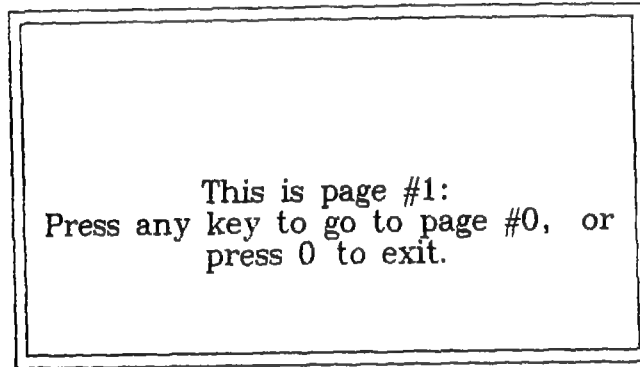
تعمل خاصية الصفحات مع الموائمات *EGA* ؛ *VGA* وكذلك الموائم وحيد اللون *Herules* فقط . كما أن الخاصية لا تعمل بكفاءة مع كل الأطوار ؛ وقد استخدمنا هنا ملف القيادة *EGA* . والطور *EGAHI* لتشغيل البرامج . ولا بأس بتجربة الأطوار الأخرى .

وفي البرنامج التالى سوف نستخدم صفحتين للرسم ، الصفحة رقم 0 والصفحة رقم 1 . وسوف يتم التبديل بينهما بصورة تكرارية كلما ضغطت على أحد الأزرار ؛ فإذا ضغطت على زر الرقم "صفر" انتهى البرنامج . وفيما يلي شاشة كل من الصفحتين رقم 0 ، ورقم 1 .



شكل (٢٣)

شاشة الصفحة رقم 0



شكل (٢٤) شاشة الصفحة رقم 1

وكما نرى أن الصفحتين تختلفان عن بعضهما البعض سواء في المحتويات أو في مواصفات بنط الكتابة المستخدم .

والعمل مقسم — في هذا البرنامج — ما بين ثلاث دوال :

● الدالة الرئيسية : لعرض الصفحات من خلال حلقة تكرارية لا نهائية وحتى يتم الضغط على الرقم 0 .

● الدالة page0 : لإعداد الرسومات والكتابة في الصفحة رقم 0 .

● الدالة page1 : لإعداد الرسومات والكتابة في الصفحة رقم ١ .

وفيما يلي نص البرنامج :

```

/* Program 6-5.cpp */
// Multiple pages
#include <graphics.h>
#include <stdlib.h>
#include <conio.h>
#include <iostream.h>
// Prototypes:
void page1(int,int);
void page0(int,int);
main()
{
    int driver = EGA;
    int mode=EGAHI;
    int errorcode;
    initgraph(&driver, &mode, "d:\\tc\\bgi");
// Read result of initialization:
    errorcode = graphresult();
    if (errorcode != grOk) {
        cout <<"Graphics error:"
             << grapherrormsg(errorcode) << endl
             << "Make sure the path to graphics "
             << "drivers is correct." << endl
             << "Press any key to exit";
        getch();
        clrscr();
        exit(1);
    }
    int HalfWidth = getmaxx()/2;
    int HalfHeight = getmaxy()/2;
// Construct pages:
    page1(HalfWidth,HalfHeight);
    page0(HalfWidth,HalfHeight);
// Display pages:
    for(;;) {
        if (getch() == '0')
            break;
        else
            setvisualpage(1);
        if (getch() == '0')
            break;
        else
            setvisualpage(0);
    }
    closegraph();
    return(0);
}

```

شکل (۲۵)


```
//
// Construct page#0:
void page0(int x,int y)
{
    setactivepage(0);
    ellipse(x,y,0,360,x,y);
    settextstyle(3,0,4);
    int z = textheight("A");
    settextjustify(CENTER_TEXT, CENTER_TEXT);
    outtextxy(x,y, "This is page #0.");
    outtextxy(x,y+z, "Press any key to go\
to page #1, or");
    outtextxy(x,y+2*z, "press 0 to exit.");
}
//
// Construct page#1:
void page1(int x,int y)
{
    setactivepage(1);
    rectangle(10,10,x*2-10,y*2-10);
    settextstyle(1,0,4);
    int z = textheight("A");
    settextjustify(CENTER_TEXT, CENTER_TEXT);
    outtextxy(x,y, "This is page #1:");
    outtextxy(x,y+z, "Press any key to go\
to page #0, or");
    outtextxy(x,y+2*z, "press 0 to exit.");
}
```

شكل (٢٦)

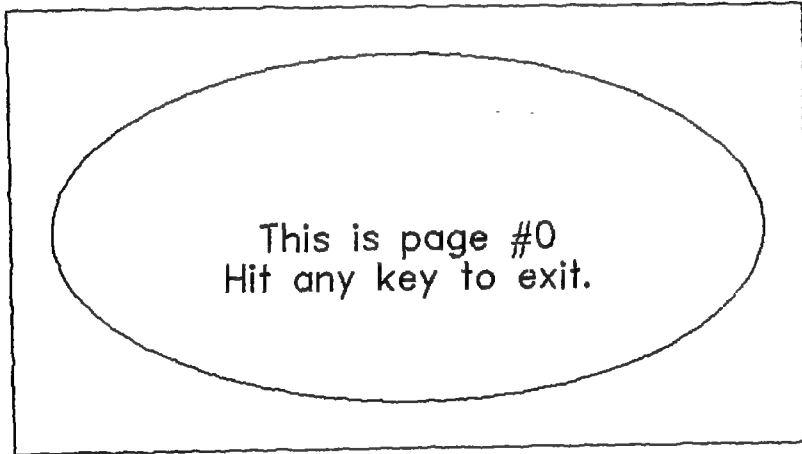
وكما نرى أن الدالة الرئيسية تحتوى على عمليتين أساسيتين هما بناء محتويات الصفحة باستخدام الدالتين "page0"، "page1". كما تحتوى على الحلقة التكرارية اللانهائية لعرض الصفحات باستخدام الدالة setvisualpage. أما الدالتان "page0"، "page1"، فهما تبدآن باستدعاء الدالة "setactivepage" حتى يتم تسجيل الرسم والكتابة في ذاكرة الصفحة المعنية.

ومن الجدير بالذكر أن عملية تبديل الصفحات ليست مجرد رسم لمشهدين بطريقة متتابعة فهي أسرع من ذلك بكثير لأن عملية الرسم قد تمت في الحقيقة مرة واحدة وتم تسجيل الصفحات في الذاكرة.

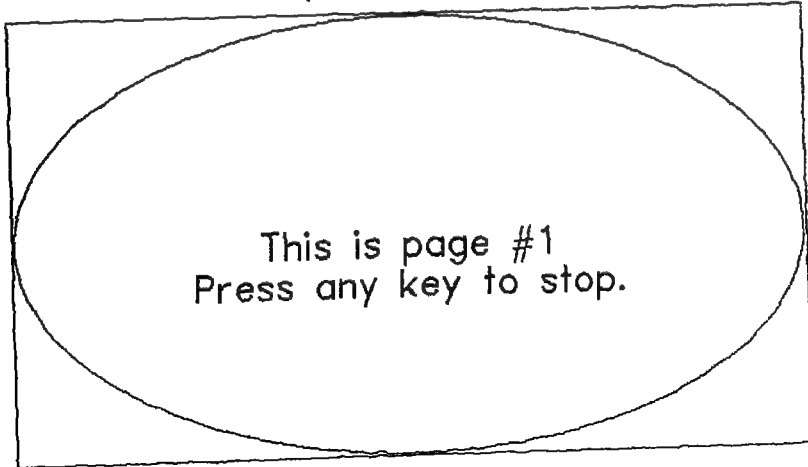
برمجة الأشكال المتحركة بتغيير الصفحات

وتعتبر عملية تبديل الصفحات إحدى الطرق في برمجة الألعاب والأشكال المتحركة على الشاشة .

وفي المثال التالي سوف نشاهد على الشاشة شكلاً يضاوياً نابضاً يتمدد وينكمش مع تغيير محتويات النص المكتوب بداخله كل مرة فإذا ضغطت على أى زر من الأزرار انتهى البرنامج .



شكل (٢٧)



شكل (٢٨)

وهذا هو البرنامج .

```

/* Program 6-6.cpp */
// Page animation
#include <graphics.h>
#include <stdlib.h>
#include <conio.h>
#include <iostream.h>
main()
{
    int driver = EGA;
    int mode=EGAH;
    int errorcode;
    initgraph(&driver, &mode, "d:\\tc\\bgi");
// Read result of initialization:
    errorcode = graphresult();
    if (errorcode != grOk) {
        cout <<"Graphics error:"
            << grapherrormsg(errorcode) << endl
            << "Make sure the path to graphics "
            << "drivers is correct." << endl
            << "Press any key to exit";
        getch();
        clrscr();
        exit(1);
    }
// Select page#1:
    setactivepage(1);
    settextstyle(3,0,4);
    settextjustify(CENTER_TEXT, CENTER_TEXT);
    int x = getmaxx()/2;
    int y = getmaxy()/2;
    int z = textheight("A");
// Draw an ellipse on page #1:
    ellipse(x, y, 0, 360, x, y);
    outtextxy(x, y, "This is page #1");
    outtextxy(x, y+z, "Press any key to stop.");
// Select page #0:
    setactivepage(0);
// Draw an ellipse on page #0:
    ellipse(x, y, 0, 360, x * 0.9, y * 0.8);
    settextstyle(3,0,4);
    outtextxy(x, y, "This is page #0");
    outtextxy(x, y+z, "Hit any key to exit.");

```

شكل (٢٩)

```
// Start an infinite loop,
// to select pages for display.
for(;;) { حلقة تكرارية لانهاية
    for(int i=0; i <= 1; i++) {
        setvisualpage(i);
        if(kbhit()) {
            closegraph();
            exit(0);
        }
        for(long k=1; k<=1000000; k++);
    }
}
```

شكل (٣٠)

ويتكون هذا البرنامج من دالة واحدة هي الدالة الرئيسية . وقد تم اختيار وبناء الصفحتين 1 ، 0 على التوالي ثم تم عرض الصفحات بطريقة متتابعة من خلال حلقة تكرارية لانهاية تنتهى عند الضغط على أى زر .

وقد تم بداخل الحلقة اللانهائية إنجاز ثلاث خطوات :

- اختيار صفحة للمشاهدة من خلال حلقة (من 0 إلى 1) باستخدام الدالة `setvisualpage` .
 - اختبار إذا ما كان المستخدم قد ضغط على أى زر باستخدام الدالة `kbhit` .
 - الدخول فى حلقة تأخير بهدف تأخير المنظر المرسوم على الشاشة . وبدون هذه الحلقة يكون التغيير سريعاً جداً .
- ومع ذلك فلو كنت تستخدم كومبيوتر محدود السرعة فعليك بتعديل قيمة الحد الأقصى للحلقة (100000) .

الموجز

[١] فى هذا الباب قد تعرفنا بمهارات متعددة تساعدنا على الاستفادة من بيئة الرسم استفادة كاملة . فقد عرفنا كيفية التعرف على الأخطاء المحتملة التى يمكن ان تحدث فى بيئة الرسم . وقد انشأنا روتيناً خاصاً للتعرف على نجاح عملية الانتقال إلى نسق رسم . وسوف نضيف هذا الروتين إلى برامجنا التالية جميعاً .

[٢] كما عرفنا كيفية إنشاء النوافذ فى نسق الرسم وكيفية التعامل معها وقياس إحداثياتها .

[٣] عرفنا أيضاً كيفية استخدام صفحات الذاكرة فى إنشاء أكثر من شاشة واحدة للرسم ، وكيفية التنقل ما بين الصفحات .

وقد كان هذا هو مدخلنا إلى تمثيل الحركة وبث الحياة فى المشاهد الجامدة . وفى الأبواب المقبلة لنا مع تمثيل الحركة جولات أخرى .

[٤] فيما يلى ملخص بالدوال الجديدة التى التقينا بها فى هذا الباب واستخدام كل منها .



دالة التعرف على نوع الخطا

```
#include <graphics.h>
int far graphresult(void);
```

دالة الحرفي المعبر عن نوع الخطا

```
#include <graphics.h>
char *far grapherrormsg(int errorcode);
```

دالة إنشاء نافذة في نسق الرسم

```
#include <graphics.h>
void far setviewport(int left, int top,
                    int right, int bottom,
                    int clip);
```

دالة التوصل إلى معلومات النافذة

```
#include <graphics.h>
void far getviewsettings (
    struct viewporttype far *viewport);
```

منشأ معلومات النافذة

```
#include <graphics.h>
struct viewporttype {
    int left;
    int top;
    int right;
    int bottom;
    int clip;
};
```

دالة مسح النافذة

```
#include <graphics.h>
void far clearviewport(void);
```

دالة مسح الشاشة (شاملة النوافذ)

```
#include <graphics.h>
void far cleardevice(void);
```

دالة إعادة النوافذ إلى الوضع سابق التعريف

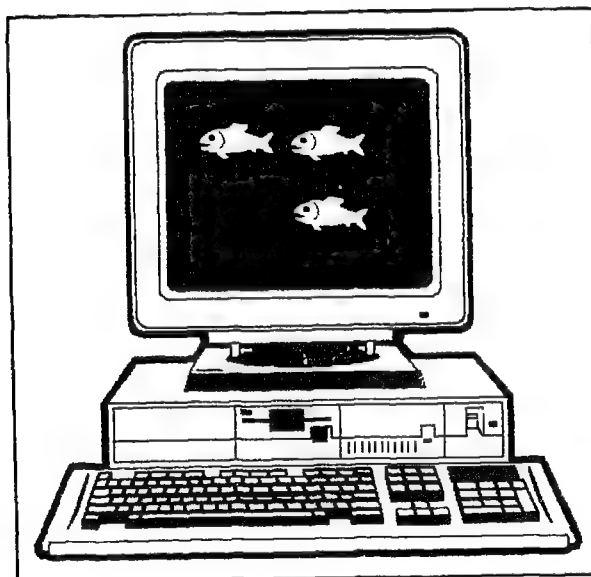
```
#include <graphics.h>
void far graphdefaults(void);
```

دالة اختيار وعرض صفحة الرسم

```
#include <graphics.h>
void far setactivepage(int page);
void far setvisualpage(int page);
```

الباب السابع

المحاكاة والأشكال الحية *animation*



مفتتح

نحن الآن فى نهاية الرحلة ، وفى أيدينا من أدوات الرسم ما يكفى لبرمجة ما نحتاج إليه . ولعله قد حان الوقت للدخول فى برامج الحركة .

وتتضمن برامج الحركة فى هذا الباب رسم خرائط الأعمدة الإحصائية بصورة حية ، وتحريك الرسومات والنصوص عبر الشاشة ؛ كما تتضمن الأشكال الخفاقة التى تتغير ألوانها بسرعة . وفى خلفية كل مثال من هذه الأمثلة تكنيك خاص من تكنيكات الحركة التى تستخدم فى برامج الألعاب الكومبيوترية على وجه الخصوص .

(٧ - ١) خرائط الأعمدة الحية

إن استخدام الرسم فى إنشاء خرائط الأعمدة يفيد فى تمثيل البيانات الإحصائية ؛ وهو لا يحتاج إلى مهارات خاصة فى الرسم فهو تطبيق مباشر على استخدام الدالة bar .

ومع ذلك فيمكننا باستخدام بعض المهارات أن نرسم الأعمدة بصورة حية بحيث نرى كل عمود وهو يصعد من قاعدته إلى قمته درجة بدرجة .

وفى المثال التالى سوف نرسم خرائط الأعمدة المثلة لبيانات المصفوفات التالية :

```
int left[9]= {50,110,170,230,290,350,410,470,530};
int right[9]={100,160,220,280,340,400,460,520,580};
int top[9]= {240,190,150,110,60,80,140,160,90};
```

شكل (١)

إن كل مصفوفة من المصفوفات الثلاث تحتوى على تسعة أرقام يمثل كل منها بيانات عمود من تسعة أعمدة .

أما المصفوفة الأولى فهى تحتوى على الإحداثى الأفقى الذى يبدأ عنده كل عمود (left) .

والمصفوفة الثانية فهى تمثل الإحداثى الأفقى الذى ينتهى عنده كل عمود (right) .

والأرقام بهاتين المصفوفتين مجرد أرقام تقديرية تناسب الشاشة VGA ويزيد كل رقم فيها عن سابقه بمقدار 60 .

أما المصفوفة الثالثة فهى تعبر عن قمم الأعمدة أى عن البيانات الفعلية المراد تمثيلها .

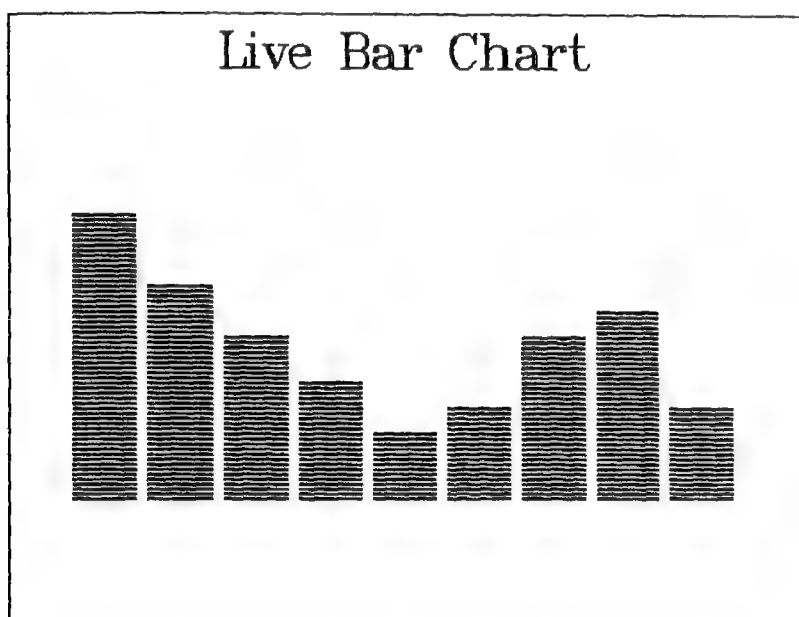
وينقصنا بالطبع رقم ثالث حتى يتم رسم العمود وهو الارتفاع الرأسى الذى يبدأ

منه العمود أو قاعدة العمود (bottom) ، وسوف نعتبر هذا الرقم ثابتاً لجميع الأعمدة وسوف نحدده بقيمة تعادل 80% من ارتفاع الشاشة أى :

```
int bottom = getmaxy()*0.8;
```

وتتم عملية الرسم بتقسيم العمود الواحد إلى مجموعة من البلوكات بحيث يتم رسمها واحداً تلو الآخر (فوق بعضها البعض) .

والشكل التالى يوضح الرسم الناتج الذى نرغب الحصول عليه من البيانات الموضحة ونلاحظ فيه أن ارتفاعات الأعمدة تتناسب مع أرقام مصفوفة البيانات . "top"



شكل (٢)

```
/* Program 7-1.cpp */
// Live Bar Chart
#include <graphics.h>
#include <stdlib.h>
#include <conio.h>
```

```

#include <iostream.h>
#include <dos.h>
const int NUM_OF_STEPS = 20;
void makebar(int, int, int, int);
main()
{
    int driver = DETECT;
    int mode;
    int errorcode;
    char title[] = "Live Bar Chart";
    // Data of the barchart (9 columns):
    // array of left(s), increment=60
    int left[9]={50,110,170,230,290,350,410,470,530};
    // array of top(s)= real amplitude of data
    int top[9]={240,190,150,110,60,80,140,160,90};
    // array of left(s) increment=60
    int right[9]={100,160,220,280,340,400,460,520,580};
    initgraph(&driver, &mode, "d:\\tc\\bgi");
    // Read result of initialization:
    errorcode = graphresult();
    if (errorcode != grOk) {
        cout << "Graphics error:"
            << grapherrormsg(errorcode) << endl
            << "Make sure the path to graphics "
            << "drivers is correct." << endl
            << "Press any key to exit";
        getch();
        clrscr();
        exit(1);
    }
    settextjustify(CENTER_TEXT, CENTER_TEXT);
    settextstyle(TRIPLEX_FONT, HORIZ_DIR, 4);
    outtextxy(getmaxx()/2, textheight("A"), title);
    setfillstyle(LINE_FILL, WHITE);
    // Define the bottom based on screen dimensions:
    int bottom = getmaxy()*0.8;
    // Call the animation function:
    for (int i=0; i<=8; i++)
        makebar(left[i], top[i], right[i], bottom);
}

```

شكل (٣)

الجزء الأول من البرنامج الأول

```

    getch();
    closegraph();
    return(0);
}
void makebar(int left, int top,
             int right, int bottom)
{
    int step = top / NUM_OF_STEPS;
    for (int i=0; i< NUM_OF_STEPS; i++) {
        bar(left, bottom-step*i, right, bottom);
        delay(10);
    }
}

```

شكل (٤)

الجزء الثانى والأخير من البرنامج الأول

مناقشة البرنامج : (وفقاً لأرقام الخطوات الموضحة بالرسم)

[١] تم فى بداية البرنامج إعلان عدد البلوكات التى يتكون منها العمود الواحد (٢٠ بلوك) باستخدام الثابت المسمى :

NUM - OF - STEPS

[٢] تم بعد ذلك إعلان المصفوفات الثلاثة :

left, top, right

التي تحتوى على الأرقام الممثلة للأعمدة .

[٣] أما البيان bottom الممثل للقاعدة فقد تم إعلانه بعد الانتقال إلى نسق الرسم .

[٤] تمت كتابة عنوان الخريطة بالدالة outtextxy وذلك بعد تحديد البنط والحجم واتجاه الكتابة .

[٥] تمت عملية الرسم باستدعاء الدالة "makebar" التى تستخدم البارامترات الأربعة الممثلة لإحداثيات أركان العمود .

[٦] أما الدالة "makebar" نفسها فيأتى تعريفها فى نهاية الملف . وهى تقوم بحساب خطوة الرسم "step" وذلك بقسمة ارتفاع العمود على عدد البلوكات .

[٧] تم بعد ذلك عملية الرسم من خلال حلقة تكرارية ونلاحظ فيها أن ارتفاع العمود المرسوم فى كل خطوة يزيد عن سابقه وذلك باستخدام التعبير :

bottom - step*i

[٨] ظهرت هنا دالة جديدة للتأخير الزمنى وهى الدالة delay التى تم تعريفها فى الملف "dos.h" وهى تأخذ بارامتراً واحداً يعبر عن زمن التأخير المطلوب بالملي ثانية . وفى البرنامج قد استخدمنا تأخيراً قدره 10 ميلي ثانية . وهذه هى عينة الدالة :

```
#include <dos.h>
void delay(unsigned milliseconds);
```

شكل (٥)

[٩] يعيب هذا البرنامج أن الأرقام فيه مُحددة تحديداً كاملاً وهذا يجعله صالحاً للتطبيق باستخدام الشاشة VGA فقط ولو أنك غيرت ملف القيادة إلى EGA أو CGA (حيث تتغير أبعاد الشاشة) فسوف تجد أن الرسم يحتل . فوق ذلك فإن البرنامج الجيد لابد أن يكون متمتعاً بالمرونة الكافية التى تجعله صالحاً لتمثيل أية بيانات فى أى عدد من الأعمدة . أما عملية حساب بدايات ونهايات الأعمدة فيجب أن يتم حسابها على أساس البيانات الجارى رسمها .

وفى البرنامج التالى نقدم صورة أفضل لنفس البرنامج وقد فضلنا أن نرجمها حتى لا تشغلنا التفاصيل عن المنطق الرئيسى للبرنامج .

```

/* Program 7-2.cpp */
// Generic live Bar Chart
#include <graphics.h>
#include <stdlib.h>
#include <conio.h>
#include <iostream.h>
#include <dos.h>
const int NUM_OF_STEPS = 20;
const int NUM_OF_COLS = 9;
const int LEFT_MARGIN = 50;
const int RIGHT_MARGIN = 50;
const int DIST_BET_COLS = 10;
const int DELAY_TIME = 10;
void makebar(int, int, int, int);
//
main()
{
    int driver = DETECT;
    int mode, errorcode;
    int counter;
    char title[] = "Live Bar Chart";
    // array of amplitudes:
    int top[NUM_OF_COLS] =
        {240,190,150,110,60,80,140,160,90};
    initgraph(&driver, &mode, "d:\\tc\\bgi");
    // Read result of initialization:
    errorcode = graphresult();
    if (errorcode != grOk) {
        cout << "Graphics error:"
            << grapherrormsg(errorcode) << endl
            << "Make sure the path to graphics "
            << "drivers is correct." << endl
            << "Press any key to exit";
        getch();
        clrscr();
        exit(1);
    }
    settextjustify(CENTER_TEXT, CENTER_TEXT);
    settextstyle(TRIPLEX_FONT, HORIZ_DIR, 4);
    outtextxy(getmaxx()/2, textwidth("A"), title);
    setfillstyle(LINE_FILL, WHITE);

```

شكل (٦)

الجزء الأول من البرنامج الثاني

```
// Define the bottom based on screen dimensions:
int bottom = getmaxy()*0.9;
int left[NUM_OF_COLS];
int right[NUM_OF_COLS];
int left_incr=
    (getmaxx()-LEFT_MARGIN-RIGHT_MARGIN)/NUM_OF_COLS;
int width=left_incr-DIST_BET_COLS;
left[0]=LEFT_MARGIN;
right[0]=LEFT_MARGIN+width;
for (counter=1; counter <= NUM_OF_COLS-1;
    counter++) {
    left[counter]=left[counter-1]+ left_incr;
    right[counter]=left[counter]+ width;
}
// Call the animation function:
for (counter=0; counter <= NUM_OF_COLS-1; counter++)
    makebar(left[counter], top[counter],
        right[counter], bottom);
getch();
closegraph();
return(0);
}
void makebar(int left, int top,
    int right, int bottom)
{
    int step = top / NUM_OF_STEPS;
    for (int i=0; i< NUM_OF_STEPS; i++) {
        bar(left,bottom-step*i,right,bottom);
        delay(DELAY_TIME);
    }
}
```

شكل (٧)

الجزء الثانى والأخير من البرنامج الثانى

مناقشة البرنامج

[١] فى هذا البرنامج تم تعريف عدة ثوابت فى بداية البرنامج وهى :

عدد الأعمدة NUM_OF_COLS = 9;
 الهامش الأيسر LEFT_MARGIN = 50;
 الهامش الأيمن RIGHT_MARGIN = 50;
 المسافة بين الأعمدة DIST_BET_COLS = 10;
 زمن التأخير DELAY_TIME = 10;

وقد تم استخدام هذه الثوابت بدلاً من الأرقام . فإذا أردت استخدام مجموعة أخرى من البيانات تشكل في مجموعها ٧ أعمدة بدلاً من تسعة فعليك بتغيير قيمة الثابت NUM-OF-COLS ؛ وبطبيعة الحال فإن عدد عناصر المصفوفة "top" سوف يكون ٧ فقط في هذه الحالة .

[٢] كما تم تقسيم الخط الأفقي إلى وحدات صغيرة يتوقف عددها على عدد الأعمدة وعلى مقدار الهامش الأيمن ومقدار الهامش الأيسر . وقد استخدمنا المتغير "left-incr" لتسمية هذه الوحدات .

[٣] أما اتساع العمود الواحد "width" فهو يساوى عرض الوحدة الواحدة "left-incr" مطروحاً منه المسافة بين الأعمدة .

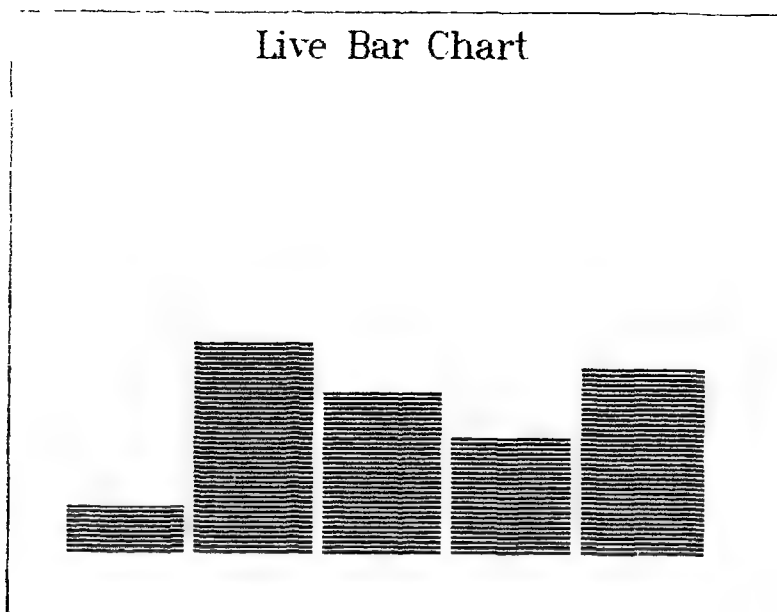
[٤] تم شحن الإحداثي left للعمود الأول بمقدار الهامش LEFT-MARGIN . وكذلك تم شحن الإحداثي right للعمود الأول بنفس القيمة مضافاً إليها اتساع العمود .

ومن خلال حلقة تكرارية تم شحن جميع مواقع الأعمدة التالية بدءاً من العمود الثاني .

[٥] يمكنك أيضاً إضافة المزيد من التعديلات بحيث تتحكم في مقياس الرسم بحسب طور الرسم المستخدم .

والشكل التالى يوضح تنفيذ البرنامج باستخدام مصفوفة أعداد مكونة من خمسة عناصر فقط ونلاحظ أن اتساعات الأعمدة قد تغيرت بحيث يملأ الرسم صفحة الشاشة .





شكل (٨)

getimage

putimage

(٧ - ٢) القص واللصق

من أهم الوسائل المستخدمة في تحريك الرسومات على الشاشة هما
الدالتان المترافقتان :

getimage

putimage

وتستخدم الدالة الأولى في نسخ الصورة المحددة بمساحة معينة على الشاشة وتخزينها في الذاكرة ؛ وتستخدم الدالة الثانية في استرجاع الصورة من الذاكرة ورسمها على الشاشة في الموضع المطلوب . لذلك فإن أحد وسائل تحريك الأشكال هي قص الرسم من مكانه ولصقه في مكان آخر ، وبتكرار العملية بالسرعة المناسبة يبدو الشكل كما لو كان متحركاً .

والدالة *getimage* تأخذ العينة الآتية :

```
#include <graphics.h>
void far getimage(int left, int top,
                 int right, int bottom,
                 void far *bitmap);
```

شكل (٩)

وكما نرى من عينة الدالة أنها تأخذ خمسة بارامترات الأربعة الأولى منها تمثل إحداثيات أركان الصورة المطلوب نسخها في الذاكرة أما البارامتر الخامس فيمثل مؤشراً خالياً من الرصيد (void) إلى الحيز الذى تشغله الصورة في الذاكرة .

وعندما تختزن الصورة في الحيز المخصص لها فإن الكلمة (word) الأولى تختص بتخزين اتساع الصورة أما الكلمة الثانية فتختص بتخزين الارتفاع . ويستخدم بقية الحيز في تخزين الصورة نفسها .

ملاحظة

الكلمة (word) تساوى ٢ بايت

وتأخذ عينة الدالة putimage الصورة الآتية :

```
#include <graphics.h>
void far putimage(int left, int top,
                 void far *bitmap,
                 int op);
```

شكل (١٠)

ونرى في عينة هذه الدالة أنها تستخدم أربعة أدلة ؛ الأول والثاني يمثلان النقطة التى نرغب فى أن نضع الصورة عندها على الشاشة . وبالطبع فإننا لا نحتاج إلى أكثر من إحداثى واحد فى هذه العملية .

أما البارامتر الثالث فهو المؤشر الذى يشير إلى الصورة فى الذاكرة . أما البارامتر الرابع فهو يحدد "مؤثر تركيب الصورة" الذى يتحكم فى طريقة تكوين الصورة النهائية باستخدام صورتين : الصورة المخزنة فى الذاكرة والصورة الموجودة بالفعل على الشاشة فى ذلك المكان .

وقد تم تعريف مؤثر تركيب الصورة فى المنشأ المتعدد "putimage-ops" بالملف "graphics.h" وهو يتكون من خمسة ثوابت تأخذ القيم العددية من 0 إلى 4 كما فى الجدول الآتى :

اسم الماكرو	القيمة العددية	// الاستخدام
COPY_PUT	0	● نسخ الصورة من الذاكرة إلى الشاشة .
XOR_PUT	1	● لإجراء العملية XOR ما بين الصورة فى الذاكرة والصورة الموجودة بالفعل على الشاشة .
OR_PUT	2	● لإجراء العملية OR ما بين الصورة فى الذاكرة والصورة الموجودة بالفعل على الشاشة .
AND_PUT	3	● لإجراء العملية AND ما بين الصورة فى الذاكرة ، والصورة الموجودة بالفعل على الشاشة .
NOT_PUT	4	● لنسخ الصورة النعكاسية من الذاكرة إلى الشاشة .

شكل (١١)

وسوف يلى عرض أمثلة كافية لاستخدام الدالتين `getimage` ،

`putimage` .

تحديد الحيز المطلوب من الذاكرة لتخزين الصورة

`imagesize`

إن اختزان مساحة معينة من الصورة المرسومة على الشاشة يتطلب حيزاً من الذاكرة يتوقف على طور الرسم المستخدم فكلما زادت دقة الرسم زاد مقدار الحيز المطلوب .

ولتحديد الحيز المطلوب لتخزين صورة محددة بإحداثيات معينة فإننا نستخدم الدالة `imagesize` التي تأخذ الصورة العامة الآتية :

```
#include <graphics.h>
unsigned far imagesize(int left, int top,
                      int right, int bottom);
```

شكل (١٢)

والدالة `imagesize` ، كما نرى من صيغتها ، تستخدم نفس البارامترات التي تحدد مساحة الصورة . وهي ترجع عدداً يمثل الحيز المطلوب للصورة (`unsigned int`) مقداراً بالبايت .

وعند استخدام هذه الدالة فإنه يلزم حجز مكان في الذاكرة للقيمة المترجمة منها باستخدام الدالة : `malloc` ، كما يلزم الإشارة إلى الحيز المحجوز بمؤشر خالٍ من الرصيد (`void`) .

وفيما يلي سوف نوضح بالأمثلة كيفية استخدام الدوال الثلاث السابقة .

© مثال توضيحي :

لنفرض أننا رسمنا شكلاً مربعاً عند الإحداثي `x,y` طول ضلعه 100 . يمكننا الآن أن ننسخ هذا المربع إلى المساحة الجديدة التي تبدأ عند النقطة `(x+50,y+50)` باتخاذ الإجراءات الآتية :

[١] تحديد حيز الذاكرة المناسب لاحتزان المربع :

```
size = imagesize(x, y, x+100, y+100);
```

إن المتغير `size` في هذه العبارة من النمط (`unsigned`) وهو يحتوى على الحيز المطلوب .

[٢] الإشارة إلى الحيز بمؤشر خالٍ من الرصيد :

```
ImagePointer = malloc(size);
```

وكما نرى أنه تم حجز الحيز المطلوب في الذاكرة باستخدام الدالة

malloc . ويتم تخصيص الناتج للمؤشر "ImagePointer" .
ويتم الإعلان عن هذا المؤشر في بداية البرنامج (قبل هذه العبارة) كمؤشر
خال من الرصيد (void) .

ملاحظة

كانت لغة سي تسمح بتخصيص مؤشر خال من الرصيد إلى مؤشر
من نوع مختلف ولكن سي++ لا تسمح بذلك إلا إذا استخدمنا الإسقاط
(casting) .

[٣] قراءة الرسم وتخزينه في الذاكرة :

```
getimage(x, y, x+100, y+100, ImagePointer);
```

وكما نرى في هذه العبارة أن المؤشر ImagePointer قد استخدم في
عملية تخزين الصورة الممتدة ما بين النقطة (x,y) والنقطة (x+100,y+100) .

[٤] لنسخ الصورة من الذاكرة إلى الشاشة مرة أخرى نستخدم عبارة مثل :

```
putimage(x+50, y+50, ImagePointer, AND_PUT);
```

ويستخدم البارامتران الأول والثاني في تحديد الركن الأيسر العلوى للصورة
أى (x+50,y+50) أما البارامتر الأخير فيحدد كيفية تركيب الصورة الجديدة
على المساحة الموجودة بالفعل بما تحتويه وهذا هو موضوع المناقشة التالية عن
"مؤثر تركيب الصورة" :

© المؤثر OR-PUT :

إن استخدام الماكرو OR-PUT يؤدي إلى إجراء العملية OR على كل
بكسلة من بكسلات الصورة مع كل بكسلة مناظرة لها في المساحة المُستقبلية
للصورة على الشاشة .

والعملية OR تعنى الآتى :

- 1 OR 1=1
- 1 OR 0=1
- 0 OR 1=1
- 0 OR 0=0

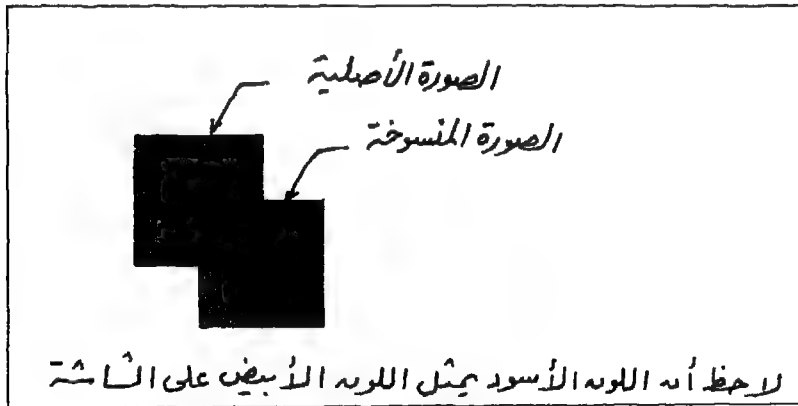
والرقم 1 هنا يعنى البكسل المضاء أى التى لونها بيض (فى حالة استخدام لون واحد للرسم) والرقم 0 يعنى البكسل غير المضاء أى التى لها نفس لون الخلفية (أسود) .

أى أنه يمكن ترجمة العلاقات الرياضية المذكورة كالتالى :

- أبيض OR أبيض = أبيض
- أبيض OR أسود = أبيض
- أسود OR أبيض = أبيض
- أسود OR أسود = أسود

أى أن الحالة التى يختفى فيها جزء من الصورة هى الحالة التى تكون فيها البكسلتان ذات لون أسود (غير مضاء) .

وبالتالى فإن تركيب الصورة باستخدام الماكرو OR—PUT يؤدى إلى الشكل التالى (لاحظ أن الألوان معكوسة فى الطباعة) :



شكل (١٣)

استخدام المؤثر OR—PUT

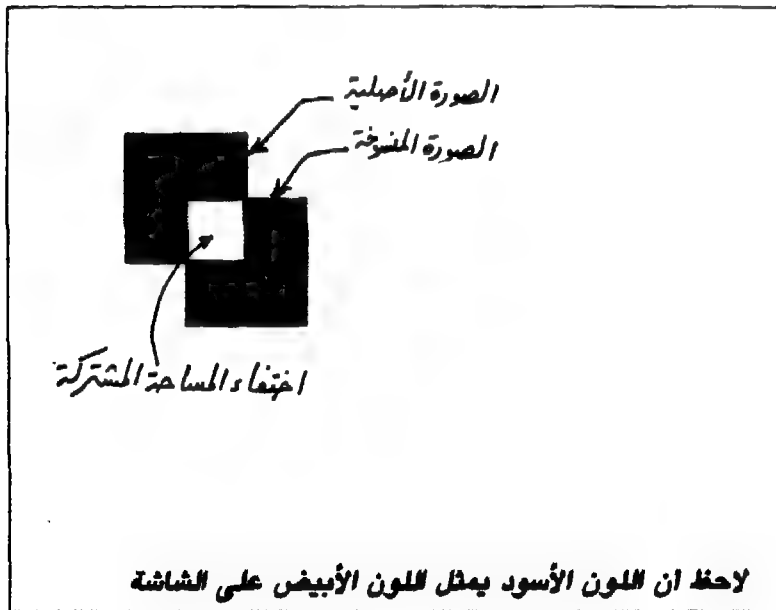
© المؤثر XOR-PUT :

أما هذا الماكرو فهو يؤدي إلى استخدام المؤثر XOR ، وهذا هو معناه :

- $1 \times \text{OR } 1 = 0$
- $1 \times \text{OR } 0 = 1$
- $0 \times \text{OR } 1 = 1$
- $0 \times \text{OR } 0 = 0$

ويمكن تلخيص جميع هذه العلاقات بأنه لو تساوى المعاملان على جانبي المؤثر XOR فإن الناتج يكون صفراً ؛ ولو اختلفا يكون الناتج 1 . وإذا طبقنا هذا المنطق على الرسم فإنه لو تشابهت البكسلتان في اللون (أبيض أو أسود) أصبحت البكسلتان الناتجة سوداء (غير مرئية) .

وهذا هو الرسم الناتج من العملية XOR-PUT :



شكل (١٤)

استخدام المؤثر XOR-PUT

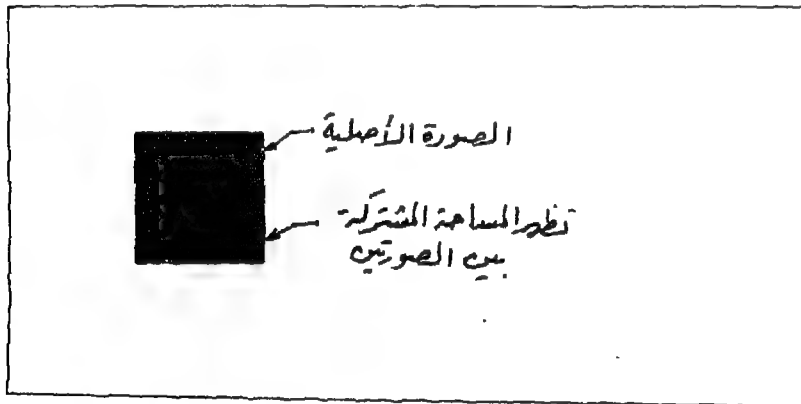
© المؤثر AND—PUT :

نعلم أن المؤثر AND لا ينتج القيمة "1" إلا إذا كان لكل من العاملين القيمة "1" بمعنى :

$$1 \text{ AND } 1 = 1$$

أما عدا ذلك فإن النتيجة تكون صفراً .

ومعنى ذلك أنك لو نسخت الشكل على مساحة خالية تماماً فلن ترى شيئاً على الشاشة لأن هذا يعادل العلاقة "1 AND 0" . أما في حالتنا هذه فإن المساحة المشتركة بين الشكلين سوف تظهر (بيضاء) أما عدا ذلك فلا . والشكل التالي يمثل نتيجة العملية AND بين الشكلين .

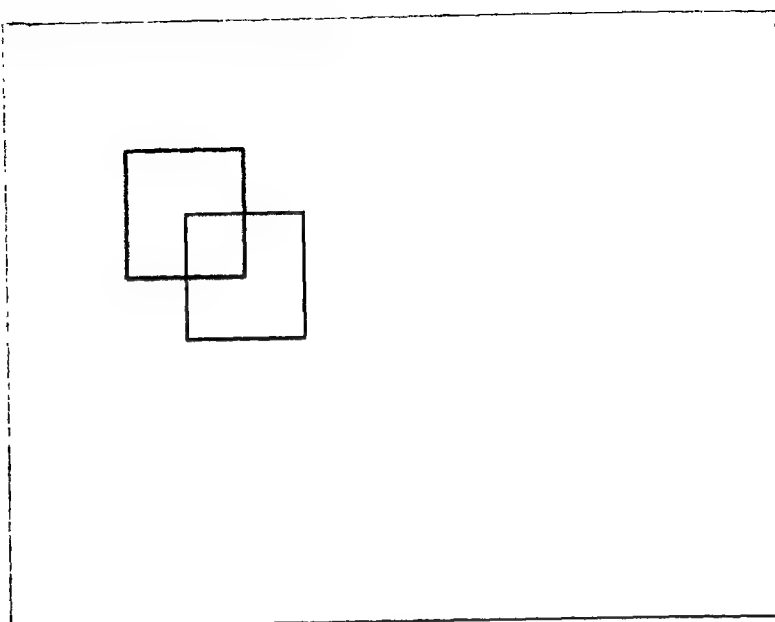


شكل (١٥)

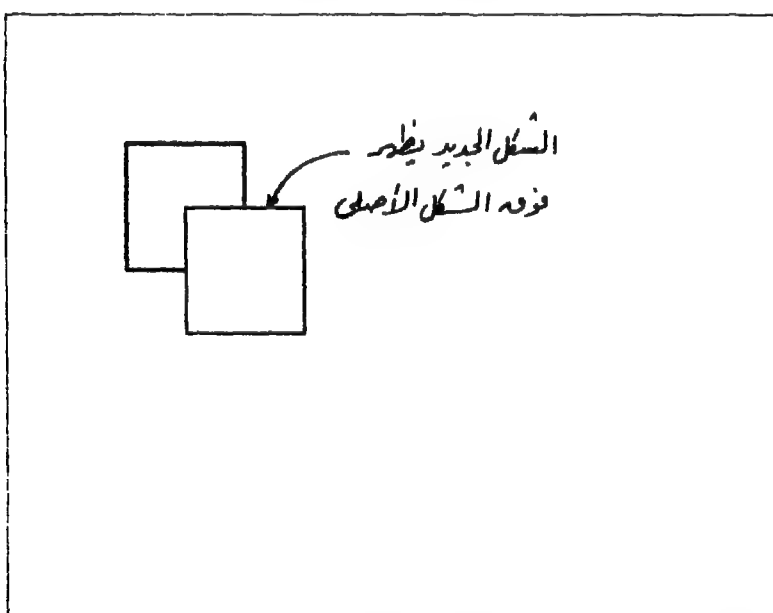
استخدام المؤثر AND—PUT

© المؤثر COPY—PUT :

يؤدي استخدام هذا المؤثر إلى نسخ الصورة من الذاكرة إلى الشاشة كما هي . والنتيجة — في مثالنا المطروح — تشبه نتيجة العملية OR—PUT تماماً . ومع ذلك فلو أننا استخدمنا شكلاً مستطيلاً بلا طلاء فسوف نرى الفارق بين المؤثر OR—PUT والمؤثر COPY—PUT . انظر الشكل التالي .



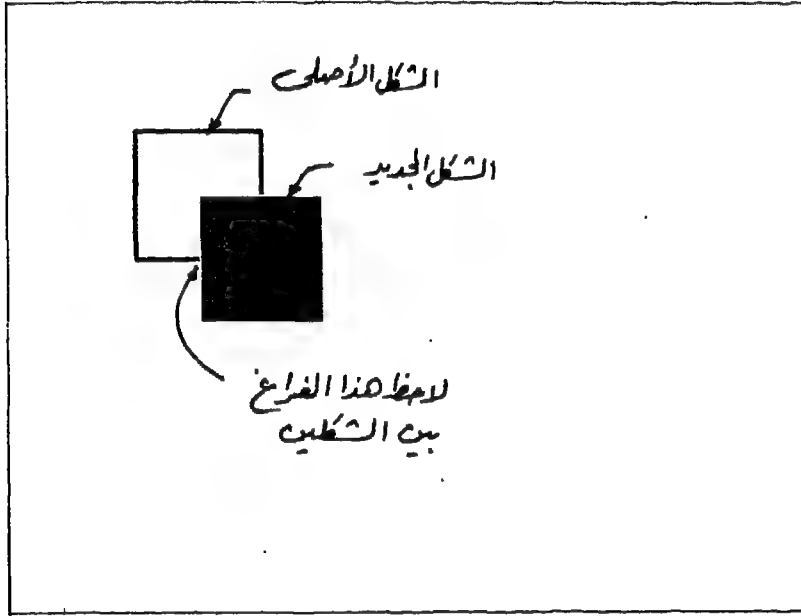
شكل (١٦)
استخدام المؤثر OR - PUT



شكل (١٧)
استخدام المؤثر COPY - PUT

© استخدام المؤثر NOT-PUT :

عند استخدام هذا المؤثر فإن الصورة الجديدة تمثل النيجاتيف للصورة الأصلية بمعنى أن كل "1" يصبح صفراً والعكس بالعكس . فلو كان الشكل الأصلي مُفرغاً ظهرت الصورة مصمتة كما في الشكل التالي :



شكل (١٨)

استخدام المؤثر NOT-PUT

ولو دقت النظر إلى الرسم لوجدت أن الشكل الجديد (النسخ) له إطار غير مرئي يؤدي إلى وجود فراغ بين الشكلين . والسبب في ذلك هو تحول الإطار الأبيض (أسود على الورق) إلى إطار أسود بلون الشاشة . ولو كان الشكل مصمتاً فسوف يختفى الربع المشترك بين الشكلين . وعند استخدام الألوان فسوف تظهر تفصيلات جديدة أثناء مزج الصور سوف نتعرض لها في الفقرات التالية .

والشكل التالي يوضح البرنامج المستخدم في إصدار الأشكال السابقة وقد استخدمنا فيه المؤثر OR-PUT وعليك باستبداله بالمؤثرات الأخرى تبعاً لتجربتها جميعاً .

```

/* Program 7-3.cpp */
// Get and put graphics
#include <graphics.h>
#include <stdlib.h>
#include <conio.h>
#include <iostream.h>
main()
{
    int driver = DETECT;
    int mode, errorcode;
    initgraph(&driver, &mode, "d:\\tc\\bgi");
    // Read result of initialization:
    errorcode = graphresult();
    if (errorcode != grOk) {
        cout << "Graphics error:"
             << grapherrormsg(errorcode) << endl
             << "Make sure the path to graphics "
             << "drivers is correct." << endl
             << "Press any key to exit";
        getch();
        clrscr();
        exit(1);
    }
    void *ImagePointer;
    unsigned int size;
    int x = 100, y = 100;
    setcolor(WHITE);
    setlinestyle(SOLID_LINE, 0, 3);
    setfillstyle(SOLID_FILL, WHITE);
    rectangle(x, y, x+100, y+100);
    floodfill(x+3, y+3, WHITE);
    size = imagesize(x, y, x+100, y+100);
    ImagePointer = malloc(size);
    getimage(x, y, x+100,
            y+100, ImagePointer);
    putimage(x+50, y+50,
            ImagePointer, OR_PUT);
    free(ImagePointer);
    getch();
    closegraph();
    return(0);
}

```

شکل (۱۹)

تدريب (٧ - ١)

يعطى البرنامج السابق شكلاً مربعاً مصغراً ، وهو لا يكفي لدراسة جميع احتمالات تركيب الصورة المنسوخة من الذاكرة على الصورة الأصلية . جرب أن تلغى الطلاء وذلك بوضع علامتى التعليق أمام السطر المشار إليه بالرقم (2) فتحصل على إطار المربع فقط .
جرب أيضاً أن تمنح إطار المربع لوناً مختلفاً مثل الأحمر وتمنح الطلاء اللون الأبيض وشاهد ما تحصل عليه من ألوان !

العمليات المنطقية على الألوان

فى الفقرة السابقة اكتفينا بلونين ، اللون الأبيض للرسم واللون الأسود للشاشة ، فكان لدينا قيمتان منطقتان فقط هما الواحد (أبيض) والصفر (أسود) أو بمعنى آخر "صحيح" و "غير صحيح" .

فإذا استخدمنا الألوان فإن البكسل يمكن أن تأخذ أوضاعاً منطقية تتعدد بتعدد الألوان . فاللون الأبيض مثلاً يحمل الرقم 15 واللون الأزرق هو الرقم 1 والأحمر هو 4 وذلك بحسب لوحة الألوان سابقة التعرف (palette) .

وعند إجراء العمليات المنطقية (OR,XOR,...) على البكسلات فى الرسم الملون فإنها يجب أن تجرى على الأرقام الثنائية الممثلة للألوان .

ولنجر الآن تطويراً على البرنامج السابق وذلك برسم إطار أحمر للشكل المربع مع الاحتفاظ باللون الأبيض للمساحة الداخلية .

عندما ننسخ الصورة من الذاكرة فى هذه الحالة فإن البرواز الأحمر عندما يوضع فوق المساحة البيضاء للصورة سوف يصدر عنهما لوناً ثالثاً يتوقف على مؤثر التركيب المستخدم . فإذا استخدمنا المؤثر XOR—PUT فإن اللون الناتج سيكون هو اللون التيركواز (cyan) وهذه هى الحسابات .

1111

0100

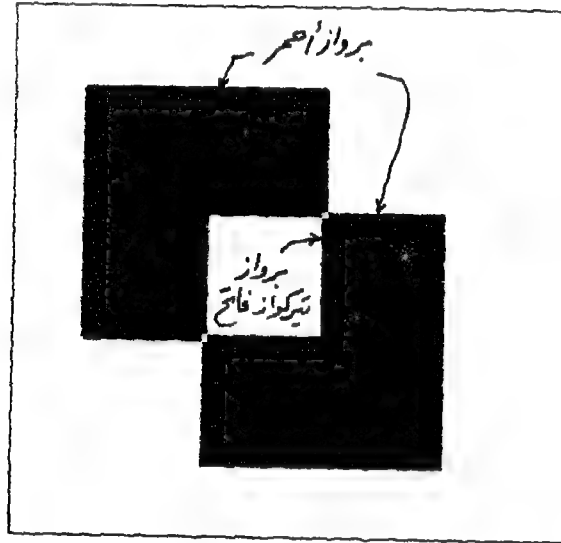
1011

● رقم اللون الأبيض 15 أى

● رقم اللون الأحمر 4 أى

● نتيجة العملية XOR

والرقم الناتج بالنظام العشري هو الرقم "11" المكافئ للون التيركواز الفاتح (LIGHT CYAN) .



شكل (٢٠)

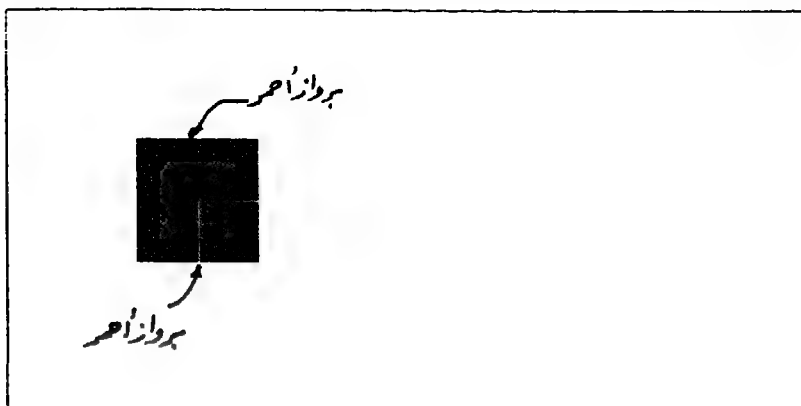
ملاحظة

إن الأرقام المستخدمة فى الحسابات السابقة هى أرقام النظام CGA وعند استخدام الأرقام الخاصة بالنظام EGA/VGA تحصل على نفس النتائج .

(انظر جدول الألوان فى الباب الأول)

ولو أنك أجريت العملية AND-PUT فسوف ترى الشكل التالى حيث

يظهر برواز الصورة المنسوخة بداخل الشكل الأصلي وبنفس اللون الأحمر .



شكل (٢١)

والسبب في ذلك هو الآتى :

1111

اللون الأبيض

0100

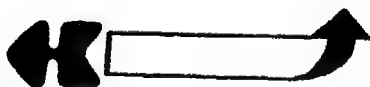
اللون الأحمر

0100

نتيجة العملية AND

أى أن الناتج هو الرقم 4 الممثل للون الأحمر .

وفيما يلى نص البرنامج المؤدى إلى الشكل السابق . أجر ما تشاء من التعديلات عليه حتى تشاهد احتمالات الألوان المختلفة .



```

/* Program 7-4.cpp */
// Get and put color graphics
#include <graphics.h>
#include <stdlib.h>
#include <conio.h>
#include <iostream.h>
main()
{
    int driver = DETECT;
    int mode, errorcode;
    initgraph(&driver, &mode, "d:\\tc\\bgi");
    // Read result of initialization:
    errorcode = graphresult();
    if (errorcode != grOk) {
        cout << "Graphics error:"
             << grapherrormsg(errorcode) << endl
             << "Make sure the path to graphics "
             << "drivers is correct." << endl
             << "Press any key to exit";
        getch();
        clrscr();
        exit(1);
    }
    void *ImagePointer;
    unsigned int size;
    int x = 100, y = 100;
    setcolor(RED);
    setlinestyle(SOLID_LINE, 0, 3);
    setfillstyle(SOLID_FILL, WHITE);
    rectangle(x, y, x+100, y+100);
    floodfill(x+3, y+3, RED);
    size = imagesize(x, y, x+100, y+100);
    ImagePointer = malloc(size);
    getimage(x, y, x+100,
            y+100, ImagePointer);
    putimage(x+50, y+50,
            ImagePointer, XOR_PUT);
    free(ImagePointer);
    getch();
    closegraph();
    return(0);
}

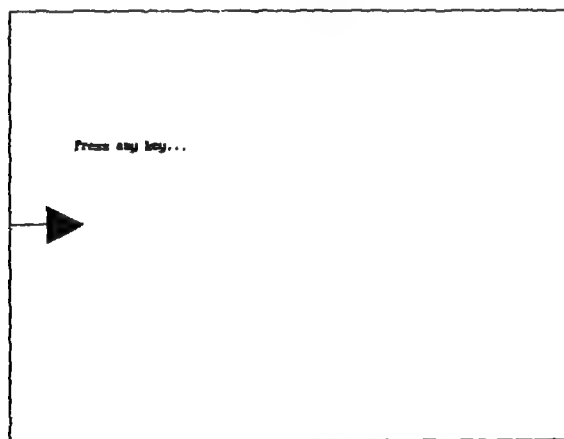
```

شكل (٢٢)

(٧-٣) رسم شكل متحرك على الشاشة

فى البرنامج التالى نستخدم الإمكانيات التى قدمناها فى الفقرات السابقة لرسم السهم الموضح بالشكل التالى ثم تحريكه على الشاشة أفقياً من يسارها إلى يمينها .

ويعمل البرنامج على خطوتين (بغرض التوضيح) فعندما يبدأ البرنامج فى التنفيذ يظهر شكل السهم الموضح بالرسم ثابتاً فى أقصى اليسار ، فإذا ضغطت على أى زر بدأ السهم حركته الأفقية بلا توقف حتى تضغط على أى زر مرة ثانية فينتهى البرنامج .



شكل (٢٣)

جرب البرنامج ثم نلتقى فى المناقشة :

```
/* Program 7-5.cpp */
// Moving arrow
#include <graphics.h>
#include <stdlib.h>
#include <conio.h>
```



```
#include <iostream.h>
#include <dos.h>
#define UNIT 20
#define DELAY 100
void DrawArrow(int x, int y);
main()
{
    int driver = DETECT;
    int mode, errorcode;
    initgraph(&driver, &mode, "d:\\tc\\bgi");
    errorcode = graphresult();
    if (errorcode != grOk) {
        cout << "Graphics error:"
             << grapherrormsg(errorcode) << endl
             << "Make sure the path to graphics "
             << "drivers is correct." << endl
             << "Press any key to exit";
        getch();
        clrscr();
        exit(1);
    }
    void *ImagePointer;
    unsigned int size;
    int x = 0, y = getmaxy()/2;
    // Draw the arrow:
    DrawArrow(x,y);
    gotoxy(10,10);
    cout << "Press any key...";
    getch();
    // Calculate the picture size:
    size = imagesize(x,
                    y - UNIT,
                    x + 4*UNIT,
                    y + UNIT);
    // Allocate the necessary memory:
    ImagePointer = malloc(size);
```

شكل (٢٤) الجزء الأول من البرنامج الخامس

```
// Save the picture in memory:
getimage(x,
        y - UNIT,
        x + 4*UNIT,
        y + UNIT,
        ImagePointer);
```

```
// Start motion:
while (!kbhit()) {
// Erase the existing image:
    putimage(x,
              y - UNIT,
              ImagePointer,
              XOR_PUT);
// Advance the x-position one unit:
    x += UNIT;
    if (x >= getmaxx())
        x = 0;
// Put an image copy in the new position:
    putimage(x,
              y - UNIT,
              ImagePointer,
              XOR_PUT);
    delay(DELAY);
}
free(ImagePointer);
closegraph();
return(0);
}

// A function to draw the arrow:
void DrawArrow(int x, int y) دالة الرسم
{
    setfillstyle(SOLID_FILL, RED);
    moveto(x,y);
// Draw the arrow:
    linerel(4*UNIT, 0);
    linerel(-2*UNIT, -1*UNIT);
    linerel(0, 2*UNIT);
    linerel(2*UNIT, -1*UNIT);
// Fill lower triangle with red color:
    int f1 = x + 2.5*UNIT;
    int f2 = y + 0.5*UNIT;
    floodfill(f1, f2, WHITE);
// Fill upper triangle with red color:
    f2 = f2 - UNIT;
    floodfill(f1, f2, WHITE);
}
```

شكل (٢٥)

الجزء الثاني والأخير من البرنامج الخامس

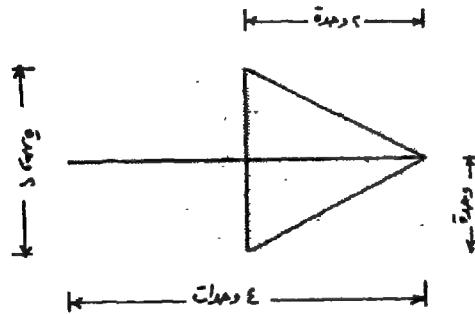
مناقشة البرنامج

ملاحظة : تتبع أرقام الخطوات فى البرنامج

① أولاً : رسم السهم :

[١] قبل البدء فى تحريك الشكل علينا أن نرسمه بطريقة دقيقة تساعدنا على التعرف على موقعه وأبعاده فى أى لحظة . والطريقة المباشرة التى تنجز هذا العمل هى رسم الشكل فى صورة وحدات كأنه مرسوم على ورقة مربعات . والرسم التالى يوضح مثلاً لذلك حيث نرى السهم مرسوماً فى مساحة طولها ٤ وحدات وعرضها وحدتان .

وقد تم فى البرنامج تحديد وحدة الرسم بالثابت "UNIT" ومنحناه القيمة 20 كما تم إعلان عينة الدالة المستخدمة فى رسم المثلث "DrawArrow" .



شكل (٢٦)

[٢] نرى فى نهاية ملف البرنامج تفصيلات دالة رسم السهم وهى تتكون أساساً من مجموعة من الخطوط تبدأ من الموقع (x,y) الذى تم إمراره إلى الدالة كبارامترات .

ويبدأ الخط الأول بخطوة مقدارها ٤ وحدات $(4 \cdot \text{UNIT})$ يمينا ، ثم يتجه

الخط الثانى إلى الشمال الغربى بإزاحة قدرها "٢- وحدة" أفقياً ووحدة واحدة رأسياً (إلى أعلى) . بعد ذلك يهبط الخط رأسياً إلى أسفل بمقدار وحدتين ؛ ثم يتجه إلى الشمال الشرق بإزاحة قدرها وحدتين أفقياً ووحدة واحدة رأسياً (إلى أعلى) .

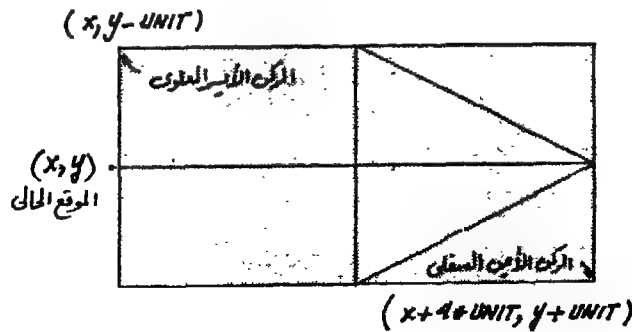
[٣] يتم بعد ذلك طلاء داخلية المثلثين باللون الأحمر وذلك بالتحرك إلى نقطة مناسبة بداخل كل مساحة ثم استخدام الدالة floodfill لملء لمثلث باللون الأحمر وحتى حدوده البيضاء .

© عملية القص واللصق (الحركة) :

تتم عملية الحركة هنا بمسح السهم المرسوم (القص) ثم إعادة رسمه على بعد وحدة واحدة من المكان السابق (اللصق) . وهذه هى الخطوات :

[٤] بعد روتين الأخطاء مباشرة تم إعلان المتغيرات اللازمة وشحن الموقع الابتدائى (x,y) ليكون عند أقصى اليسار في منتصف الشاشة تماماً . ثم تمت عملية الرسم المبدئى للسهم في هذا الموقع يعقبا استخدام الدالة getch لتعليق البرنامج لحين الضغط على أى زر .

[٥] تأتى بعد ذلك عملية تقدير سعة الذاكرة المطلوبة لتخزين صورة السهم وهذا يستلزم قراءة إحداثيات كل من الركنين الأيسر العلوى والأيمن السفلى للمستطيل الذى يحتوى على الشكل . وبلاستعانة بالشكل التالى يمكن استنتاج الإحداثيات التى تم استخدامها .



شكل (٢٧)

[٦] تم بعد ذلك استخدام الدالة `getimage` باستخدام نفس البارامترات المستخدمة مع الدالة `imageSize` وذلك لحفظ الصورة في الذاكرة بالفعل والإشارة إلى عنوانها هناك بالمؤشر `ImagePointer` المعلن عنه في بداية البرنامج .

[٧] يبدأ بعد ذلك روتين تحريك الشكل من خلال حلقة تكرارية (`while`) تنتهى عند الضغط على أى زر .

وتبدأ العملية بمسح الصورة الموجودة على الشاشة وذلك بوضع نسخة مماثلة فوقها تماماً ، ويتولى مؤثر التركيب `XOR—PUT` عملية المسح (كما سبق بالأمثلة) .

[٨] يتم بعد ذلك تقديم الموقع الحالى بمقدار وحدة واحدة (`UNIT`) أى ما يوازي ربع طول السهم .

ولا ننسى إضافة شرط لإعادة السهم إلى يسار الشاشة ($x=0$) كلما وصل إلى أقصى اليمين .

[٩] يتم نسخ الصورة من الذاكرة إلى الموقع الجديد باستخدام الدالة `putimage` وباستخدام المؤثر `XOR—PUT` (ويجوز استخدام مؤثرات أخرى طالما أنه قد تم مسح الصورة القديمة) .

بعد لصق الصورة تستخدم الدالة `delay` لإحداث تأخير زمنى قدره ٢٠ ميلي ثانية بموجب الثابت `DELAY` المعلن عنه في بداية البرنامج .

ومن الجدير بالذكر أن التحكم في قيمة هذا الثابت يؤدي إلى إسرار الحركة أو إبطائها ولذلك يجب إعطاؤه القيمة المناسبة في حالة إذا ما كانت الحركة بطيئة على شاشتك .

[١٠] في نهاية البرنامج تستخدم الدالة `free` لإتاحة الذاكرة المحجوزة للصورة وهى الدالة المقابلة لدالة `malloc` التى استخدمناها في حجز الذاكرة .

فلاش

هناك حدود للمساحات التى يمكن قصها ولصقها باستخدام الدوال `putimage` ، `getimage` . فهذه الدوال قد صُممت مبدئياً لتخزن مساحات من الصورة فى حيز لا يزيد عن 64 كيلوبايت .

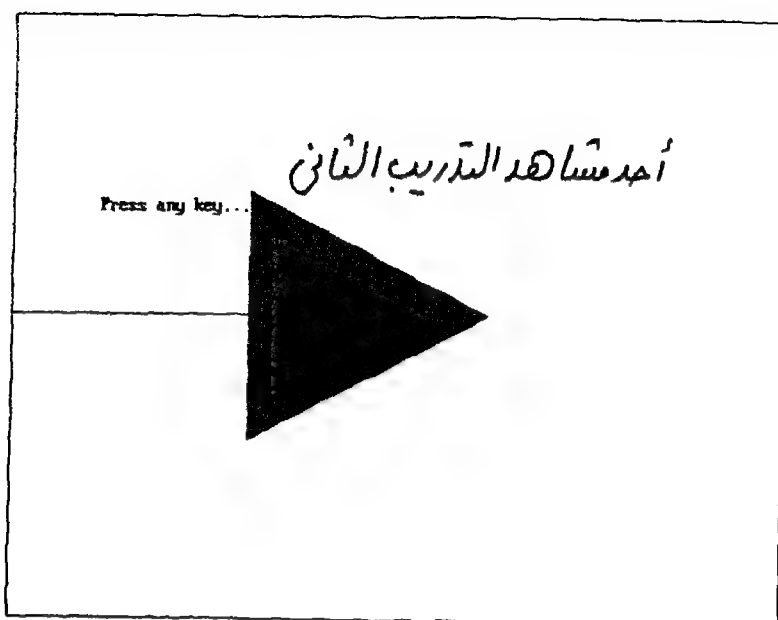
فعلى سبيل المثال لو أنك استخدمت درجة الدقة 640×200 فإنه يتعذر تخزين مساحة الشاشة كلها لأن ذلك يحتاج إلى حيز من الذاكرة أكبر من 64KB . وكلما زادت درجة الدقة كلما قل الحد الأقصى للمساحة التى يمكن قصها ولصقها .

تدريب (٧ - ٢)

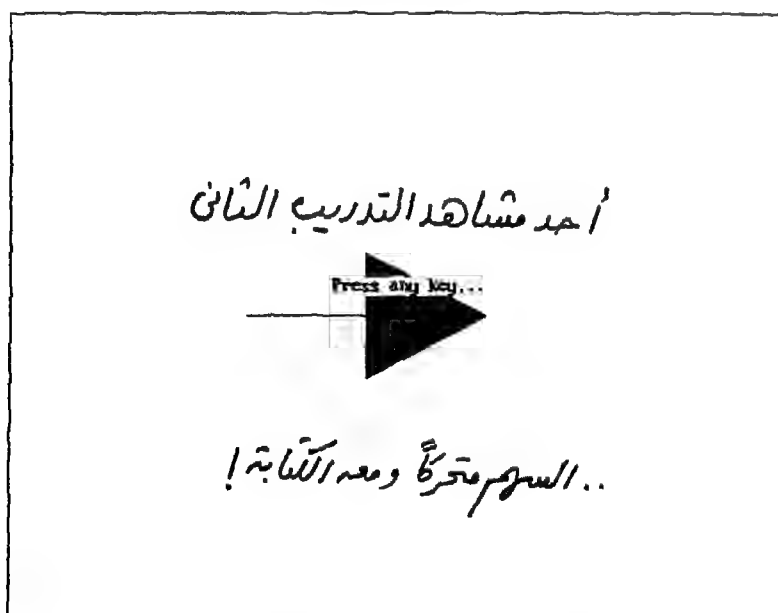
يمكنك إجراء عدة تعديلات على البرنامج السابق تساعدك على فهم المنطق الأساسى الذى يتم به رسم وتحريك الأشكال . فيمكنك تكبير وتصغير الشكل المرسوم بمجرد تغيير مساحة الوحدة (UNIT) . جرب وشاهد .

كما يمكنك أن تجعل الكتابة الموجودة على الشاشة تتحرك مع السهم المتحرك وذلك بالتحكم فى المساحة المقصودة من المشهد . جرب وشاهد .

يمكنك أيضاً استخدام هذا البرنامج كأساس لعملية تحريك أى شكل وذلك بتغيير منطق دالة الرسم `DrawArrow` بحيث تحتوى نفس المساحة على شكل جديد .



شكل (٢٨)



شكل (٢٩)

(٧ - ٤) تحريك نص على الشاشة (Moving Text)

بنفس المبدأ يمكنك أن تكتب نصاً متحركاً على الشاشة فيما عدا أن العبارة المكتوبة في مساحة معينة على الشاشة يسهل تحديد إحداثياتها باستخدام دوال قياس اتساع النص textwidth وارتفاعه textheight .

وفي البرنامج التالي نكتب العبارة "Moving Text" على الشاشة بحيث تتحرك باستمرار من اليسار إلى اليمين أعلى الشاشة ، فإذا وصلت إلى أقصى اليمين بدأت دورة جديدة على سطر جديد حتى تصل إلى قاع الشاشة . وعندما تختفى من قاع الشاشة تبدأ رحلة جديدة من القمة إلى القاع .

```
/* Program 7-6 */
// Moving text
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <iostream.h>
#include <dos.h>
#define DELAYTIME 80
#define VTAB 1.5
#define FONTSIZE 4
//
main()
{
    int driver = DETECT;
    int mode, errorcode;
    initgraph(&driver, &mode, "d:\\tc\\bgi");
    errorcode = graphresult();
    if (errorcode != grOk) {
        cout << "Graphics error:"
            << grapherrormsg(errorcode) << endl
            << "Make sure the path to graphics "
            << "drivers is correct." << endl
    }
```



```

        << "Press any key to exit";
        getch();
        clrscr();
        exit(1);
    }
    int x=0, y=0;
    char *string = "Moving Text";
    unsigned int size;
    void *ImagePointer;
    int Maxx = getmaxx();
    int Maxy = getmaxy();
    settextstyle(TRIPLEX_FONT, HORIZ_DIR, FONTSIZE);
    moveto(x,y);
    // Display the string:
    outtext(string);
    // Calculate image size:
    int X = textwidth(string);
    int Y = VTAB * textheight(string);
    size = imagesize(x, y, x+X, y+Y);
    ImagePointer = malloc(size);
    // Save image:
    getimage(x, y, x+X, y+Y, ImagePointer);

```

شكل (٣٠)

الجزء الأول من البرنامج السادس

```

// Start motion loop:
while (!kbhit()) {
    putimage(x, y, ImagePointer, XOR_PUT);
    x += textwidth("I");
    if (x > Maxx) {
        y += Y;
        x = 0;
    }
    if (y > Maxy-Y) y=0;
    putimage(x, y, ImagePointer, XOR_PUT);
    if (x < Maxx-X)
        delay(DELAYTIME);
    else
        delay(0.1*DELAYTIME);
}

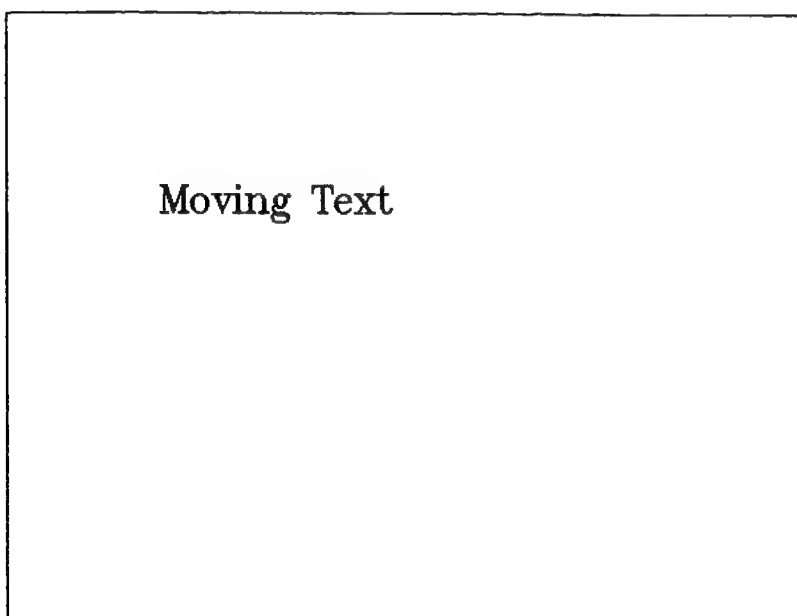
```

```

    }
    free(ImagePointer);
    closegraph();
    return(0);
}

```

شكل (٣١)
الجزء الثانى والأخير من البرنامج السادس



شكل (٣٢)

مناقشة البرنامج

[١] هناك تعديل جد فى هذا البرنامج على كيفية استخدام زمن التأخير
DELAYTIME . فبالرغم من تحديد الزمن كعدد ثابت فى بداية البرنامج :

```
#define DELAYTIME 80
```

مع ذلك فإنه قد تم استخدام جزء صغير من هذا الزمن عندما يصل النص إلى النهاية اليمنى للشاشة . والهدف من ذلك ألا يختفى النص لفترة طويلة عن عينيك أثناء رحلته من اليمين إلى اليسار في الخلفية :

```
if (x < Maxx-X)
    delay(DELAYTIME);
else
    delay(0.1*DELAYTIME);
```

شكل (٣٣)

[١] تم تعريف ثابتين في بداية البرنامج هما حجم البنت (4) والبياضة الرأسية (Vertical Tab) ومقدارها (1.5) .

```
#define VTAB 1.5
#define FONTSIZE 4
```

شكل (٣٤)

أما البنت فيمكنك تغييره وبالتالي تغيير مساحة الرسم كلها بمجرد تغيير قيمة الثابت FONTSIZE . وأما البياضة الرأسية VTAB فهي معامل يُضرب في ارتفاع النص لحساب الإزاحة الرأسية التي تأخذ مجراها عندما ينتقل النص من سطر إلى سطر . وقد قدرنا هذا المعامل هنا بمقدار 1.5 ولك أن تغير هذا الرقم وتشاهد تأثيره على الحركة .

```
int X = textwidth(string);
int Y = VTAB * textheight(string);
```

شكل (٣٥)

ومن الجدير بالذكر أن البياضة الرأسية تدخل في تقدير مساحة الصورة المطلوب تخزينها في الذاكرة سواء مع الدالة imagesize أو الدالة getimage . ومن البديهي أنه كلما زاد مقدار البياضة الرأسية "Y" فإن الحركة تصبح أبطأ .

```
size = imagesize(x, y, x+X, y+Y);
ImagePointer = malloc(size);
getimage(x, y, x+X, y+Y, ImagePointer);
```

شكل (٣٦)

[٣] لاحظ أيضاً أن ترحيل النص إلى اليمين بمقدار اتساع حرف واحد وهو حرف "I" وهذا هو العامل الثالث الذى يؤثر على سرعة الحركة .

تدريب (٧-٣)

اكتب برنامجاً لتحريك عبارة ما على الشاشة رأسياً من القمة إلى القاع ، ومن اليسار إلى اليمين فى هيئة أعمدة رأسية .

(٧-٥) الحركة بتغيير لوحة الألوان

setpalette

مررنا فى الباب الأول بالألوان مروراً عابراً لكنه كان مع ذلك كافياً للغرض . ولقد عرفنا كيفية تحديد لوحة الألوان (Palette) للنظام القديم CGA لكننا لم نتعرض للوحة الألوان للنظم المتطورة EGA ، VGA واكتفينا بلوحة الألوان سابقة التعريف التى تحتوى على 16 لوناً ، تبدأ باللون الأسود وتنتهى باللون الأبيض . ومع ذلك فإن هناك توليفة من الألوان تحتوى على 64 لوناً يمكن منها اختيار مجموعة الألوان . ويتم ذلك باستخدام الدالة *setpalette* التى تأخذ الصورة الآتية :

```
#include <graphics.h>
void far setpalette(int colornum,
                  int color);
```

شكل (٣٧)

وتستخدم الدالة في تغيير لون واحد من ألوان اللوحة "colornum" إلى لون جديد "color". فعلى سبيل المثال يمكنك تغيير اللون رقم صفر (اللون الأسود) إلى اللون رقم 4 (اللون الأحمر) بالعبرة :

```
setpalette(0,4);
```

فلو كان هناك رسم باللون الأحمر على الشاشة فإنه عقب هذه العبارة سوف يختفي لأن الشاشة كلها ستصبح حمراء .

ويجوز مع كارت الرسم EGA/VGA تغيير بعض أو كل ألوان اللوحة ولكنه مع الكارت CGA لا يمكن تغيير إلا لون الخلفية فقط باستخدام هذه الدالة .

ويسرى التغيير الناتج عن هذه الدالة على جميع الألوان الموجودة على الشاشة فوراً .

ملاحظة

لا تستخدم هذه الدالة مع الكارت IBM-8514 . ولتجربة هذه الدالة نفذ البرنامج التالي الذى يطبع على الشاشة 15 عبارة ملونة بالألوان المتاحة فى لوحة الألوان وعندما تضغط على أى زر يتم تغيير كل لون إلى اللون التالى له .

Color: 1
Color: 2
Color: 3
Color: 4
Color: 5
Color: 6
Color: 7

Color: 9
Color: 10
Color: 11
Color: 12
Color: 13
Color: 14
Color: 15

Press a key to change color..

شكل (٣٨)

```
/* Program 7-7.cpp */
// Changing palette colors
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <iostream.h>
main()
{
    int driver = DETECT;
    int mode, errorcode;
    initgraph(&driver, &mode, "d:\\tc\\bgi");
    errorcode = graphresult();
    if (errorcode != grOk) {
        cout << "Graphics error:"
            << grapherrormsg(errorcode) << endl
            << "Make sure the path to graphics "
            << "drivers is correct." << endl
            << "Press any key to exit";
    }
    getch();
    clrscr();
    exit(1);
}
```

```

    }
    settextstyle(TRIPLEX_FONT, HORIZ_DIR, 2);
    int Height;
    int x = 100, y = 10;
    char String[80];
    Height = 1.2 * textheight("A");
    // Display the default colors:
    for (int color=1; color<=getmaxcolor(); color++) {
        setcolor(color);
        sprintf(String, "Color: %d", color);
        outtextxy(x, y, String);
        y += Height;
    }
    outtextxy(x, y+Height,
              "Press a key to change color..");
    getch();
    // Change colors one by one:
    for (color=1; color<=getmaxcolor(); color++) {
        setpalette(color, color+1);
        getch();
    }
    closegraph();
    return(0);
}

```

شكل (٣٩)

أما في البرنامج التالي فإننا نستخدم خاصية تغيير الألوان في تحويل الشكل المرسوم إلى شكل نابض بالحياة ، حيث نبدأ برسم النجمة الموضحة بالشكل التالي مع طلائها بستة ألوان مختلفة ، فإذا ضغطت على أى زر فإن الألوان تتبدل باستمرار وبصورة عشوائية .

Press any key...



تنفيذ البرنامج الثامن
شكل (٤٠)

```

/* Program 7-8.cpp */
// Blinking Star
#include <graphics.h>
#include <stdlib.h>
#include <conio.h>
#include <iostream.h>
#include <dos.h>
#define UNIT 100
#define DELAYTIME 10
void DrawStar(int x, int y);
main()
{
    int driver = DETECT;
    int mode, errorcode;
    initgraph(&driver, &mode, "d:\\tc\\bgi");
    errorcode = graphresult();
    if (errorcode != grOk) {
        cout << "Graphics error:"
             << grapherrormsg(errorcode) << endl
             << "Make sure the path to graphics "
             << "drivers is correct." << endl
             << "Press any key to exit";
        getch();
        clrscr();
        exit(1);
    }
    int x = getmaxx()/2, y = getmaxy()/2;
    int color=getmaxcolor();
    // Draw the star:
    DrawStar(x,y);
    settextstyle(TRIPLEX_FONT, HORIZ_DIR, 4);
    settextjustify(CENTER_TEXT, CENTER_TEXT);
    outtextxy(x, getmaxy()/8, "Press any key...");
    getch();
    for (;;) {
        if (!kbhit()) {
            color=random(15)+1;
            setpalette(color,random(15)+1);
            delay(DELAYTIME);
        }
        else {
            closegraph();
            return(0);
        }
    }
}

```

شکل (۴۱)


```
// A function to draw the star:
void DrawStar(int x, int y)
{
    int color = getmaxcolor();
    moveto(x,y-UNIT);
    // Draw the star:
    linerel(UNIT, 2*UNIT);
    linerel(-2*UNIT, -1.5*UNIT);
    linerel(2*UNIT, 0);
    linerel(-2*UNIT, 1.5*UNIT);
    linerel(UNIT, -2*UNIT);
    // Fill
    setfillstyle(SOLID_FILL, color);
    floodfill(x, y, color);
    setfillstyle(SOLID_FILL, color-1);
    floodfill(x, y-0.7*UNIT, color);
    setfillstyle(SOLID_FILL, color-2);
    floodfill(x+0.7*UNIT, y-0.4*UNIT, color);
    setfillstyle(SOLID_FILL, color-3);
    floodfill(x-0.7*UNIT, y-0.4*UNIT, color);
    setfillstyle(SOLID_FILL, color-4);
    floodfill(x+0.4*UNIT, y+0.1*UNIT, color);
    setfillstyle(SOLID_FILL, color-5);
    floodfill(x-0.4*UNIT, y+0.1*UNIT, color);
}
```

شكل (٤٢)

مناقشة البرنامج

[١] يتم رسم النجمة باستخدام الدالة DrawStar التي جاءت عيبتها في أول البرنامج . ونستخدم هنا نفس المبدأ الذي استخدمناه في رسم "السهم" حيث قسمنا مساحة الرسم إلى أربع وحدات (٢ × ٢) . وفي دالة الرسم ، جاءت دوال الرسم أولاً تليها دوال الطلاء .

[٢] أما عملية تغيير الألوان فقد تمت باستخدام الدالة **setpalette** من خلال حلقة تكرارية لا نهائية تنتهى عند الضغط على أحد الأزرار .
وحتى يكون تغيير الألوان عشوائياً فقد تم الاستعانة بالدالة **random** .
والدالة **random** تأخذ الصيغة العامة الآتية :

```
#include <stdlib.h>
int random(int num);
```

شكل (٤٣)

والعدد العشوائى الناتج من هذه الدالة يقع بين الصفر ، (num-1) . وفى برنامجنا الحالى قد تم إضافة "1" إلى قيمة الدالة حتى يكون العدد الناتج بين "1" ، "15" أى أنه يغطى جميع الألوان فيما عدا اللون الأسود (لون الخلفية) . ويبدأ اللون فى البرنامج بأقصى قيمة له **getmaxcolor** ولكننا أثناء عملية تغيير اللون نختار لوناً عشوائياً من الألوان الخمسة عشر ثم نغير قيمته إلى قيمة عشوائية جديدة .

تدريب (٧ - ٤)

أضف إلى البرنامج السابق الكود اللازم لتحريك النجمة أفقياً عبر الشاشة فى دورات مستمرة وذلك علاوة على تغيير الألوان .

تطبيقات اخرى للوحة الألوان

لا حدود للتطبيقات التى يمكننا أن نستخدم فيها تغيير لوحة الألوان لكى نبث الحياة فى مشهد ما . فعلى سبيل المثال يمكننا تمثيل حركة الموج على شاطئ البحر وانعكاسات الشمس عليه بمجموعة من الألوان المتابعة .

كما يمكننا تمثيل صواريخ الاحتفالات التي تنطلق إلى السماء وتبعثر في كل اتجاه وبكل لون . إنها مسألة تعتمد أساساً على خيال المبرمج وتتطلب بعض الجهد لتحقيق المشهد المطلوب .

وفي البرنامج التالي نقدم تطبيقاً آخر لهذا المبدأ حيث نضيف إضافة جديدة إلى البرنامج (٤ - ١٢) الذي رسمنا فيه الجبل والبحر وخط الساحل ، حيث نغير لون خط الساحل الأبيض إلى اللون السماوي (التيروكواز) ثم الأزرق الفاتح ثم الأزرق ، وذلك باستخدام فترات تأخير مناسبة لكل لون . وبهذا فإن المشهد النهائي يحاكي حركة الموج التي نشاهدها من مسافة بعيدة على شاطئ هادئ .

ملاحظة

في هذا البرنامج تم حذف الإطار الأبيض الذي يحيط بالمشهد .

```
/* Program 7-9.cpp */
// Live seashore
#include <graphics.h>
#include <conio.h>
#include <math.h>
#include <stdlib.h>
#include <time.h>
#include <dos.h>
#include <iostream.h>
//
#define SIZE 1000
#define MAXPERTURB 6
//
double Frac_Array[SIZE];
void Fractalize(int,int,double,double);
void Limits(int,int,int,int,double,double);
void Draw_It(int);
main()
{
//
```

```

int driver = DETECT;
int mode, errorcode;
initgraph(&driver, &mode, "d:\\tc\\bgi");
// Read result of initialization:
errorcode = graphresult();
if (errorcode != grOk) {
    cout << "Graphics error:"
        << grapherrormsg(errorcode) << endl
        << "Make sure the path to graphics "
        << "drivers is correct." << endl
        << "Press any key to exit";
    getch();
    clrscr();
    exit(1);
}
// Initialize random function:
randomize();
// Fractalize the mountain line and draw it:
setcolor(CYAN);
Limits(100,100,MAXPERTURB,0.3,50.0);
Draw_It(100);
// Fill the sky with CYAN color:
setfillstyle(SOLID_FILL,CYAN);
floodfill(1,1,CYAN);
// Fractalize the seashore line and draw it:
setcolor(WHITE);
Limits(150,150,MAXPERTURB,0.9,30.0);
Draw_It(150);

```

شكل (٤٤) الجزء الأول من البرنامج التاسع

```

// Fill the sea with BLUE color:
setfillstyle(SOLID_FILL,BLUE);
floodfill(1,getmaxy()-1,WHITE);
// Emulate the motion of waves on the shore:

```

```

for (;;) {
    if (!kbhit()) {
        setpalette(WHITE,LIGHTBLUE);
        delay(random(900)+500);
    }
    if (!kbhit()) {
        setpalette(WHITE,CYAN);
        delay(random(500)+200);
    }
    if (!kbhit()) {
        setpalette(WHITE,BLUE);
        delay(random(900)+500);
    }
    else {
        closegraph();
        return(0);
    }
}
}
//
// A function to process the limits
// of fractalization:
void Limits(int y1,int y2,
            int MaxPerturb,
            double DecayFctr,double FrScale)
{
    int First, Last;
    double FrRatio, FrDecay;
    First=0;
    Last=(int)pow(2.0,(double)MaxPerturb);
    Frac_Array[First]=y1;
    Frac_Array[Last]=y2;
    FrRatio=1.0/pow(2.0,DecayFctr);
    FrDecay=FrScale*FrRatio;
    Fractalize(First,Last,FrDecay,FrRatio);
}
//

```

شكل (٤٥)
الجزء الثاني من البرنامج التاسع

```
// The Fractalization process:
void Fractalize(int y1,int y2,
               double FrDecay,double FrRatio)
{
    int Middle;
    double NewFrScale;
    Middle=(y1+y2)/2;
    if(Middle != y1 && Middle !=y2) {
        Frac_Array[Middle]=
            (Frac_Array[y1]+Frac_Array[y2])/2 +
            (double)((random(16)-8))/8.0*FrDecay;
        NewFrScale=FrDecay*FrRatio;
        Fractalize(y1,Middle,NewFrScale,FrRatio);
        Fractalize(Middle,y2,NewFrScale,FrRatio);
    }
}
//
// Drawing Fractalized lines:
void Draw_It(int y)
{
    int xinc, L;
    L= (int)pow(2.0,(double)MAXPERTURB);
    xinc=getmaxx()/L*3/2;
    moveto(0,y);
    for(int i=0, x=0; i<L; i++, x+=xinc)
        lineto(x,(int)Frac_Array[i]);
}
```

شكل (٤٦)

الجزء الثالث والأخير من البرنامج التاسع

● (٦-٧) القص واللصق فى نسق الكتابة
 gettext
 puttext

كما فى نسق الرسم ، فإنه يمكنك القص واللصق أيضاً فى نسق الكتابة
 باستخدام الدالتين :

```
int gettext(int left, int top,
            int right, int bottom,
            void *destin);
int puttext(int left, int top,
            int right, int bottom,
            void *source);
```

شكل (٤٧)

ومن الجدير بالذكر أن الإحداثيات الواردة في عينات الدوال هي إحداثيات شاشة النصوص التي تبدأ عند العمود الأول والصف الأول (1,1) وتنتهى عند (80,25) .

والدالة gettext تؤدي إلى اختزان اللبنة الموجودة في المستطيل الذي يبدأ عند العمود والصف "top"، "left"، وينتهى عند العمود والصف "right"، "bottom"، في حيز الذاكرة المشار إليه بالمؤشر "destin" (وقد يكون هذا المؤشر مصفوفة لبنات) .

ولقراءة ٢٣ سطراً من الشاشة وتخزينها في الذاكرة فإننا نستخدم عبارة

مثل :

```
    }
    gettext(1, 1, 80, 23, Screen);
```

في هذه الحالة فإن حيز الذاكرة Screen يحتفظ بالنص المقصود لحين استرجاعه وطبعه على الشاشة باستخدام الدالة puttext .

والدالة puttext تؤدي إلى طباعة النص عند الإحداثيات المستخدمة كبارامترات للدالة ولا يشترط بالضرورة أن يطبع عند نفس الإحداثيات التي كان موجوداً بها من قبل .

أما حيز الذاكرة المطلوب فمقداره 2 بايت لكل لبنة ، حيث تخصص بايت واحدة لللبنة ولبنة أخرى للخصائص (اللون) . ومن الجدير بالذكر أن هذه الدوال تعمل في حدود الشاشة كلها وليس من خلال نافذة .

وفي البرنامج التالي نطبع على الشاشة ٢٣ سطراً ثم نخزن المحتويات في الحيز

Screen ، ونمسح الشاشة . ثم نطبع محتويات الحيز Screen مرة أخرى في نفس الإحداثيات .

والملاحظ هنا أنه قد تم تغيير اللون عدة مرات أثناء الحوار ومع ذلك فإنه عند استعادة الكتابة الأصلية التي تم اختزانها فإنها عادت بلونها الأصلي ساعة التخزين .

*****	Line #81	*****
*****	Line #82	*****
*****	Line #83	*****
*****	Line #84	*****
*****	Line #85	*****
*****	Line #86	*****
*****	Line #87	*****
*****	Line #88	*****
*****	Line #89	*****
*****	Line #10	*****
*****	Line #11	*****
*****	Line #12	*****
*****	Line #13	*****
*****	Line #14	*****
*****	Line #15	*****
*****	Line #16	*****
*****	Line #17	*****
*****	Line #18	*****
*****	Line #19	*****
*****	Line #20	*****
*****	Line #21	*****
*****	Line #22	*****
*****	Line #23	*****
Here is your stuff safe and sound! Press any key.		

شكل (٤٨)
تنفيذ البرنامج العاشر




```

/* Program 7-10.cpp */
// Get and put text
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
//
main()
{
    char Line[80];
    void *Screen = malloc(4000);
    clrscr();
    textcolor(CYAN);
    for (int i = 1; i <= 23; i++) {
        sprintf(Line, "*****\
Line %%.2d *****", i);
        cprintf("%s\r\n", Line);
    }
    gettext(1, 1, 80, 23, Screen);
    gotoxy(1, 25);
    cprintf("Press any key to clear screen...");
    getch();
    clrscr();
    gotoxy(1, 25);
    textcolor(WHITE);
    cprintf("Now press any key to get your screen\
back...");
    getch();
    textcolor(YELLOW);
    puttext(1, 1, 80, 23, Screen);
    gotoxy(1, 25);
    cprintf("Here is your stuff safe and sound!\
Press any key.");
    getch();
    return(0);
}

```

شكل (٤٩)

الموجز

[١] فى هذا الباب التقينا بمجموعة من المهارات التى تساعدنا على محاكاة الأشكال المتحركة فعرفنا كيفية تمثيل خرائط الأعمدة الإحصائية بطريقة ديناميكية . وكذلك عرفنا الدالتين :

`getimage`

`putimage`

اللتين نستخدمان معاً فى محاكاة الأشكال المتحركة وقد أطلقنا على العملية إجمالاً اسم القص واللصق .

[٢] وقد خصصنا جزءاً كبيراً من هذا الباب لطرق استخدام القص واللصق فى تحريك الأشكال على الشاشة ، كما عرفنا أيضاً كيفية تحريك النصوص على الشاشة .

[٣] عرفنا أيضاً فى هذا الباب أسلوباً جديداً لمحاكاة الأشكال المتلألئة مثل النجوم وانعكاسات الضوء على خط الساحل مع حركة الموج ؛ وذلك بتغيير لوحة الألوان باستخدام الدالة `setpalette` . وقد قدمنا عدة تطبيقات لهذه الدالة بما فى ذلك تطوير برنامج الرسم بالفراكتلات الذى بدأناه فى الباب الرابع .

[٤] التقينا أخيراً فى هذا الباب بدوال القص واللصق فى نسق الكتابة :

`gettext`

`puttext`

[٥] فيما يلى تقدم ملخصاً بالدوال التى استخدمناها فى هذا الباب .



آلة التأخير الزمني

```
#include <dos.h>
void delay(unsigned milliseconds);
```

دالة حفظ الرسم في الذاكرة

```
#include <graphics.h>
void far getimage(int left, int top,
                 int right, int bottom,
                 void far *bitmap);
```

دالة استرجاع الرسم من الذاكرة

```
#include <graphics.h>
void far putimage(int left, int top,
                 void far *bitmap,
                 int op);
```

دالة تحديد الحيز اللازم من الذاكرة

```
#include <graphics.h>
unsigned far imagesize(int left, int top,
                     int right, int bottom);
```

دالة تغيير ألوان لوحة الألوان

```
#include <graphics.h>
void far setpalette(int colornum,
                  int color);
```

دالة توليد الأعداد العشوائية

```
#include <stdlib.h>
int random(int num);
```

دالة تخزين النصوص في الذاكرة

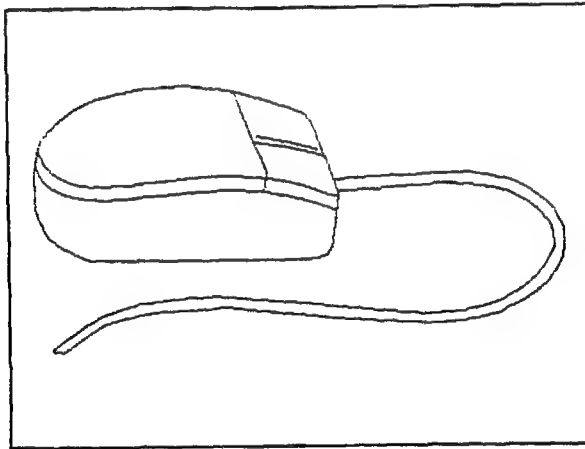
```
#include <conio.h>
int gettext(int left, int top,
           int right, int bottom,
           void *destin);
```

دالة استرجاع النصوص من الذاكرة

```
int puttext(int left, int top,
           int right, int bottom,
           void *source);
```


الباب الثامن

برمجة الفأر الإلكتروني *Mouse Programming*



تفويه

© إن البرامج الموجودة في هذا الباب تستخدم خصائص البرمجة الموجهة نحو الأهداف :

“Object Oriented Programming”

لغة سي++ . ويلزم للمبرمج أن يكون ملماً بهذه المبادئ حتى يمكنه متابعة هذا الباب .

وللاطلاع على مبادئ البرمجة الموجهة نحو الأهداف يُرجع للجزء الأول من كتابنا :

سي++/سي++ للنوافذ/البرمجة الموجهة نحو الأهداف

© يحتوى هذا الباب على مشروعات (projects) وهي تتكون من عدة ملفات تتكامل معاً لتكوين الملف التنفيذي (EXE) ، ولا تعمل هذه الملفات منفردة . هذه الملفات هي :

8-5.PRJ	يتبع المشروع	8-5.cpp
8-6.PRJ	يتبع المشروع	8-6.cpp
8-7.PRJ	يتبع المشروع	8-7.cpp
8-8.PRJ	يتبع المشروع	8-8.cpp

(٨ - ١) استخدام الفأر الإلكتروني

يتكون الفأر من عنصرين ؛ الجهاز الميكانيكي للفأر (المُعَدَّة) والبرنامج المُقيم في الذاكرة الذي يستخدم في قيادة الجهاز (mouse driver) .

وعادة فإن برنامج قيادة الفأر يتم تحميله في الذاكرة عند بدء تشغيل الكمبيوتر حيث يكون أمر تحميل البرنامج هو أحد الأوامر المتضمنة في ملف الأوامر للبدء (autoexec.bat) . ويعتبر برنامج قيادة الفأر هو المسئول عن الاتصال بين الفأر وبين سائر البرامج التي تستخدم الفأر . ويجوز استخدام الفأر في نسق الكتابة أو نسق الرسم ومع ذلك فإنه من المتوقع أن نحتاج إليه أكثر في تطبيقات الرسم .

وفي الفقرات القادمة سوف نستعرض أهم الدوال التي تستخدم في التعامل مع الفأر ومن هذه الدوال سوف نبني في النهاية فصيلة (class) للفأر تشكل في مجموعها الأدوات اللازمة للتحكم في الفأر . ولا يخفى على مبرمج لغة سي++ أهمية وجود الدوال كحزمة واحدة في فصيلة ما فهذا يجعل برنامجك التطبيقي بسيطاً ومتسلسلاً بحيث لا تختلط السطور الخاصة بالفأر بسطور البرنامج الأصلي حيث أن إعلان الفصيلة — كما نعلم — يوضع في ملف عناوين مستقل .

ويتميز أيضاً استخدام الفصيلة عن استخدام مجموعة متفرقة من الدوال بأن الفصيلة كائن حي يمكن بسهولة أن تلد فصيلة أخرى باستخدام الوراثة (inheritance) ويمكن تعديل خصائصها وإضافة إليها باستخدام تعدد الأشكال (polymorphism) .

(٨ - ٢) الفأر والمسجلات وخطوط المقاطعة (Mouse, interrupts, registers)

يتم التعامل مع الفأر عن طريق خط المقاطعة (interrupt) رقم 33h .

ملاحظة .

إن خط المقاطعة عبارة عن برنامج تابع لنظام التشغيل يقيم في الذاكرة وجاهز على الاستخدام . وإن الكثير من أوامر نظام التشغيل مثل *COPY* ، *XCOPY* ، *TIME* ، *FORMAT* تستخدم أوامر المقاطعة المبنية في نظام التشغيل .

ويتم التعامل مع خط المقاطعة عن طريق المسجلات (registers) باستخدام دالة لغة سي *int86* التي تأخذ الصورة العامة :

```
int int86(int int_number,
          union REGS *in_regs,
          union REGS *out_regs);
```

شكل (١)

والبارامترات المستخدمة في الدالة هي :

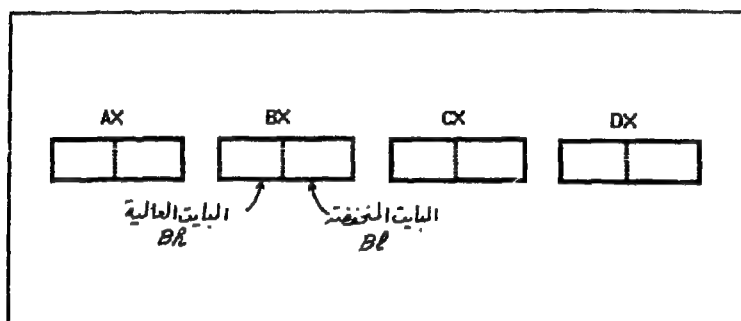
int—number رقم خط المقاطعة .

in—regs مسجلات الدخل .

out—regs مسجلات الخرج .

والشكل التالي يوضح المسجلات المستخدمة وهي *AX* ، *BX* ، *CX* ،

DX ويتسع كل مسجل لعدد ٢ بايت (كلمة) ، وقد ظهر في الرسم خط يقسم كل مسجل إلى قسمين حيث يمكن التعامل مع كل نصف على حدة ؛ وفي هذه الحالة يطلق على النصف الأيمن "البايت المنخفضة" وعلى النصف الأيسر "البايت العالية" حيث تتسع كل بايت لعدد يتراوح بين ٠ ، 255 .



شكل (٢)

وفي ملف العناوين DOS.h تم تعريف هذه المسجلات في صورة منشآت :

WORDREGS لتمثيل المسجل كمجموعة من الكلمات .

BYTEREGS لتمثيل المسجل كمجموعة من البايت .

ويرتبط هذان المنشآت في منشأ مشترك (union) بحيث يمكن التعامل مع أيهما ولكنه لا يجوز التعامل مع كليهما في نفس الوقت لأن أعضاء المنشأ المشترك (وكذلك المسجلات) تشغل نفس الحيز من الذاكرة . وهذا هو تعريف المنشأ المشترك :

```
union REGS {
    struct WORDREGS x;
    struct BYTEREGS h;
};
```

شكل (٣)

أما هذه فهي تعريفات المنشآت :

```

struct WORDREGS {
    unsigned int    ax, bx, cx, dx;
    unsigned int    si, di;
    unsigned int    cflag, flags;
};

struct BYTEREGS {
    unsigned char    al, ah, bl, bh;
    unsigned char    cl, ch, dl, dh;
};

```

شكل (٤)

ونلاحظ أن المنشأ الثاني يستخدم الحرف L للدلالة على البايت المنخفضة والحرف H للدلالة على البايت العالية في كل مسجل . أما المنشأ الأول فهو يحتوي على مسجلات أخرى بخلاف AX ، BX ، CX ، DX ولن نتعرض لها في هذا المجال . ولكي نتوصل إلى أعضاء المنشآت فإنه يلزمنا الإعلان عن هدف (object) من نفس النمط . انظر هذا المثال :

```

#include <dos.h>
main()
{
    union REGS in_regs;
    union REGS out_regs;
    in_regs.x.ax = 0;
    int86(0x33, &in_regs, &out_regs);
    return(0);
}

```

شكل (٥)

في هذا المثال تم شحن مسجل الدخول AX الموجود في المنشأ "in—regs" بالعدد صفر . وقد تم التوصل إلى المتغير AX بالعبرة :

in—regs.x.ax=0

ونذكرك بأنه يلزم استخدام اسم المنشأ المشترك متبوعاً باسم العضو (مثل

x أو h) وإذا كان العضو عبارة عن منشأ (كما هو الحال في هذا المثال) فإنه يمكن التوصل إلى عناصر العضو أيضاً بنفس الطريقة . واستخدام مؤثر النقطة هنا لازم للفصل بين المنشأ والعضو التابع له .

وبتخصيص القيمة صفر للمسجل AX فإنه يمررها إلى خط المقاطعة 33h ، والذي بدوره يقوم بوضع الفأر في حالته الابتدائية وهذا هو معنى الرقم 0 في المسجل AX ، حيث أن كل رقم يوضع في المسجل AX له معنى خاص كما سيلي .

ويحتوى ملف قيادة الفأر (mouse driver) على مجموعة من الدوال تحمل كل منها رقماً مسلسلاً ، فإذا وضعنا هذا الرقم في المسجل AX فإن الدالة المناظرة لهذا الرقم يتم استدعاؤها عن طريق خط المقاطعة 33h .

وفيما يلي نوضح أرقام الدوال التى يتضمنها ملف قيادة الفأر طراز ميكروسوفت (Microsoft Mouse) وللمزيد من الدوال يمكنك الرجوع إلى دفتر المبرمج للفأر :

“Microsoft Mouse Programmers Guide”

ويطلق على هذه الدوال إجمالاً اسم دوال الخدمة أو روتينات الخدمة للفأر .



الوظيفة	رقم الدالة (قيمة المسجل AX)
شحن الفأر بالأحوال الابتدائية للتشغيل (الإعداد) .	0
إظهار مؤشر الفأر على الشاشة .	1
إخفاء مؤشر الفأر من الشاشة .	2
إرجاع قيمة موقع الفأر على الشاشة وحالة أزراره .	3
تحريك مؤشر الفأر إلى موقع معين (x,y) .	4
إرجاع عدد مرات الضغط على زر الفأر .	5
إرجاع عدد مرات إطلاق زر الفأر .	6
تحديد نطاق حركة الفأر أفقياً .	7
تحديد نطاق حركة الفأر رأسياً .	8
تحديد نوع المؤشر المستخدم في نسق الرسم .	9
ضبط المؤشر المستخدم في نسق الكتابة .	10
قراءة عدادات الحركة للفأر .	11
إعداد روتين خط المقاطعة .	12
تشغيل خاصية محاكاة القلم الضوئي .	13
تبطيل خاصية محاكاة القلم الضوئي .	14
ضبط النسبة بين حركة الفأر وحركة المؤشر .	15
إخفاء الفأر خلال نطاق محدد .	16
إعداد بارامترات الحركة السريعة .	19
تبديل روتينات المقاطعة .	20
التعرف على حالة برنامج قيادة الفأر .	21
حفظ حالة برنامج قيادة الفأر .	22
استرجاع حالة برنامج قيادة الفأر .	23
إعداد رقم الصفحة المستخدمة بواسطة الفأر .	29
إيجاد رقم الصفحة المستخدمة بواسطة الفأر .	30

شكل (٦)

جدول دوال الخدمة للفأر (ميكروسوفت)

وفي هذا الباب سوف نستخدم بعض هذه الدوال بلغة سي++ كمقدمة لاستخدام الفأر ويمكنك استكمال الرحلة بإضافة ماتراه لازماً من هذه الدوال أو من الدوال الموجودة في دفتر برمجة الفأر .

(٨-٣) دالة لشحن الفأر بالأحوال الابتدائية

إن البرنامج الصغير الذي استخدمناه التو يتضمن دالة الخدمة رقم 0 المستخدمة في إعداد الفأر وشحنه بالأحوال الابتدائية اللازمة للتشغيل . وإذا نجحت الدالة في أداء عملها فإنها تضع القيمة (1-) في المسجل AX (بمعنى إذا كان الفأر موجوداً في الفتحة المخصصة له ، وإذا تم التعرف على برنامج قيادة الفأر) وإلا فإنها ترجع القيمة صفراً .

وفي البرنامج التالي سوف نعيد كتابة نفس الدالة في صورة دالة عضوة في الفصيلة "mouse" مع إضافة بعض روتينات الاختبار إلى دالة البرنامج الرئيسي ، هذا علاوة على إعداد نسق الرسم في نفس البرنامج .

```
/* Program 8-1.cpp */
// Initialize the mouse
#include <dos.h>
#include <stdlib.h>
#include <conio.h>
#include <graphics.h>
#include <iostream.h>
// The mouse class declaration:
// to go into the header file "mouse.h"
```

```

class mouse {
public:
    virtual int initialize(void);
};
// The mouse constants:
// to go into the header file "mouse.h"
const int RESET = 0;
main()
{
    int driver = DETECT, mode, errorcode;
    mouse MyMouse; // The mouse object
    initgraph(&driver, &mode, "d:\\tc\\bgi");
    // Graphics initialization:
    errorcode = graphresult();
    if (errorcode != grOk) {
        cout << "Graphics error:"
            << grapherrormsg(errorcode) << endl
            << "Make sure the path to graphics "
            << "drivers is correct." << endl
            << "Press any key to exit";
        getch();
        clrscr(); exit(1);
    }
    // Mouse initialization:
    if (!MyMouse.initialize()) {
        cout << "Mouse could not be initialized."
            << endl << "Press any key to exit";
        getch();
        clrscr(); exit(1);
    }
    else {
        cout << "Mouse has been initialized successfully."
            << endl << "The returned value from the "
            << "output register AX is: "
            << MyMouse.initialize();
    }
    getch();
    return(0);
}

```

شكل (٧)

البرنامج الأول - الجزء الأول

```
// First member function in the mouse class:
int mouse::initialize(void)
{
    union REGS in_regs;
    union REGS out_regs;
    // Write a value in the input register:
    in_regs.x.ax = RESET;
    // Invoke the interrupt 33h:
    int86(0x33, &in_regs, &out_regs);
    // Return the value of the output register:
    return(out_regs.x.ax);
}
```

شكل (٨)

البرنامج الأول - الجزء الثاني والأخير

مناقشة البرنامج

ينقسم البرنامج إلى دالة رئيسية (main) ودالة عضوة هي دالة إعداد الفأر . وفيما يلي من فقرات سوف نضيف المزيد من الدوال الأعضاء إلى البرنامج .

[١] تحتوي الدالة الرئيسية على إعلان فصيلة الفأر "mouse" وهي تحتوي على عضو واحد حتى الآن . وكلما أنشأنا عضواً جديداً عليك أن تضيفه إلى الفصيلة .

وعندما تكتمل أعضاء الفصيلة سوف نكتبها جميعاً في ملف للعناوين "mouse.h" كما هو المتبع .

[٢] تحتوي الدالة الرئيسية أيضاً على ثابت واحد وهو الثابت "RESET" وعندما ننشئ المزيد من الدوال سوف يزيد عدد الثوابت فنكتبها جميعاً في الملف "mouse.h".

[٣] بفرض أن الدخول في نسق الرسم قد تم بنجاح ، وأن عملية إعداد

الفأر قد تمت بنجاح فإن هذا البرنامج سوف يسفر عن الرسالة :

Mouse has been initialized successfully.

The returned value from the output register AX is: - 1

[٤] أما إذا فشلت العملية لسبب ما مثل عدم وجود جهاز الفأر أو عدم وجود برنامج قيادة الفأر في الذاكرة فإنه يسفر عن الرسالة :

Mouse could not be initialized.

Press any key to exit.

[٥] أما برنامج الدالة فنلاحظ أنه يستخدم المسجل AX مرةً للدخل ومرة للخروج . ففي الحالة الأولى نكتب فيه ثابت الإعداد RESET (وهو يساوى صفراً) . وبعد تمام العملية نقرأ قيمة المسجل التي تمثل القيمة المرتجعة من الدالة .

فإذا كانت هذه القيمة "1-" (أو TRUE) كان هذا علامة النجاح أما إذا كانت صفراً (أو FALSE) فمعنى هذا أن الدالة فشلت في إعداد الفأر .

[٦] من المهم أن تلاحظ طريقة استدعاء دالة عضوة ، فلا بد أولاً من الإعلان عن هدف من النمط mouse مثل MyMouse ثم استخدام الهدف في استدعاء الدالة مثل :

MyMouse.initialize()

(٨ - ٤) دالة إظهار مؤشر الفأر

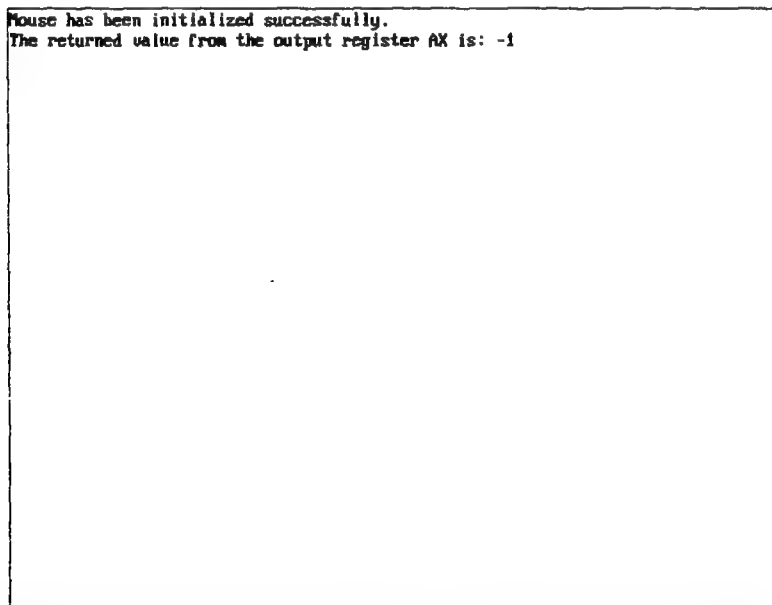
كل ما هو مطلوب منك حتى تظهر مؤشر الفأر على الشاشة ، أن تكتب الرقم 1 (وهو رقم دالة إظهار المؤشر - في الجدول) في المسجل AX ، ثم تستدعي الدالة int86 كالمعتاد . وهذه هي الدالة العضوة الجديدة التي تحتوى على هذا المنطق :


```
// Second member function
// to show the mouse cursor.
void mouse::show_cursor(void)
{
    union REGS in_regs;
    union REGS out_regs;
    // Write the value 1 into AX:
    in_regs.x.ax = SHOW;
    int86(0x33, &in_regs, &out_regs);
}
```

شكل (٩)

وعندما تستدعى هذه الدالة من خلال البرنامج الرئيسى فإنها تؤدي إلى إظهار مؤشر الفأر على الشاشة . ويأخذ المؤشر شكل رأس سهم فى نسق الرسم (ولكنه يظهر على شكل مستطيل فى نسق الكتابة) .

وبإضافة هذه الدالة إلى البرنامج السابق فإننا نحصل على الشكل التالى عند تنفيذ البرنامج :



شكل (١٠)

ومؤشر الفأر الذى يظهر على الشاشة مؤشر قابل للحركة فى جميع الاتجاهات . ومن الجدير بالذكر أنه من اللازم قبل إظهار المؤشر أن نستخدم دالة إعداد الفأر لشحنه بالأحوال الابتدائية وإلا فإن استدعاء دالة إظهار المؤشر لن يكون له أى تأثير .

وفيما يلى نص البرنامج الثانى كاملاً وقد وضعنا فى الدالة الرئيسية الإضافات التى استلزمها إضافة الدالة الجديدة .

```
/* Program 8-2.cpp */
// Initialize and show the mouse cursor
#include <dos.h>
#include <stdlib.h>
#include <conio.h>
#include <graphics.h>
#include <iostream.h>
// The mouse class declaration:
// to go into the header file "mouse.h"
class mouse {
public:
    virtual int initialize(void);
    virtual void show_cursor(void);
};
// The mouse constants:
// to go into the header file "mouse.h"
const int RESET = 0;
const int SHOW = 1;
main()
{
    int driver = DETECT, mode, errorcode;
    mouse MyMouse; // The mouse object
    initgraph(&driver, &mode, "d:\\tc\\bgi");
    // Graphics initialization:
    errorcode = graphresult();
    if (errorcode != grOk) {
        cout << "Graphics error:"
             << grapherrormsg(errorcode) << endl
             << "Make sure the path to graphics "
             << "drivers is correct." << endl
             << "Press any key to exit";
    }
}
```

```

        getch();
        clrscr(); exit(1);
    }
    // Mouse initialization:
    if (!MyMouse.initialize()) {
        cout << "Mouse could not be initialized."
              << endl << "Press any key to exit";
        getch();
        clrscr(); exit(1);
    }
    else {
        cout << "Mouse has been initialized successfully."
              << endl << "The returned value from the "
              << "output register AX is: "
              << MyMouse.initialize();
    }

```

شكل (١١) الجزء الأول من البرنامج الأول

```

    }
    MyMouse.show_cursor();
    getch();
    return(0);
}
// First member function in the mouse class:
int mouse::initialize(void)
{
    union REGS in_regs;
    union REGS out_regs;
    // Write a value in the input register:
    in_regs.x.ax = RESET;
    // Invoke the interrupt 33h:
    int86(0x33, &in_regs, &out_regs);
    // Return the value of the output register:
    return(out_regs.x.ax);
}

```



```

}
// Second member function
// to show the mouse cursor.
void mouse::show_cursor(void)
{
    union REGS in_regs;
    union REGS out_regs;
    // Write the value 1 into AX:
    in_regs.x.ax = SHOW;
    int86(0x33, &in_regs, &out_regs);
}

```

شكل (١٢) الجزء الثانى والأخير من البرنامج الأول

٨-٥ (دالة إخفاء المؤشر)

عند كتابة القيمة 2 فى مسجل الدخل AX فإن هذا يؤدي إلى استدعاء دالة إخفاء المؤشر (انظر الجدول) .

وهذا هو برنامج الدالة :

دالة إخفاء المؤشر

```

// The hide-cursor function:
void mouse::hide_cursor(void)
{
    union REGS in_regs;
    union REGS out_regs;
    // Write the value 2 into AX:
    in_regs.x.ax = HIDE;
    int86(0x33, &in_regs, &out_regs);
}

```

شكل (١٣)

وفى البرنامج الثالث سوف نضيف هذه الدالة من خلال الحوار الموضح بالشكل التالى حيث يبدأ البرنامج والمؤشر على الشاشة . وبالضغط على أحد الأزرار يختفى المؤشر وبضغط زر آخر يعود المؤشر إلى الظهور .

```
Mouse has been initialized successfully.
The returned value from the output register AX is: -1
```

```
Press a key to hide the cursor.
Press a key to restore the cursor.
Press a key to exit.
```

شكل (١٤)

```
/* Program 8-3.cpp */
// Initialize, show, and hide the mouse cursor
#include <dos.h>
#include <stdlib.h>
#include <conio.h>
#include <graphics.h>
#include <iostream.h>
// The mouse class declaration:
// to go into the header file "mouse.h"
class mouse {
public:
    virtual int initialize(void);
    virtual void show_cursor(void);
    virtual void hide_cursor(void);
};
// The mouse constants:
// to go into the header file "mouscnst.h"
const int RESET = 0;
const int SHOW = 1;
const int HIDE = 2;
main()
```

```

{
    int driver = DETECT, mode, errorcode;
    mouse MyMouse; // The mouse object
    initgraph(&driver, &mode, "d:\\tc\\bgi");
// Graphics initialization:
    errorcode = graphresult();
    if (errorcode != grOk) {
        cout << "Graphics error:"
            << grapherrormsg(errorcode) << endl
            << "Make sure the path to graphics "
            << "drivers is correct." << endl
            << "Press any key to exit";
        getch();
        clrscr(); exit(1);
    }
// Mouse initialization:
    if (!MyMouse.initialize()) {
        cout << "Mouse could not be initialized."
            << endl << "Press any key to exit";
        getch();
        clrscr(); exit(1);
    }
    else {
        cout << "Mouse has been initialized successfully."
            << endl << "The returned value from the "
            << "output register AX is: "
            << MyMouse.initialize();
    }
}

```

شكل (١٥)

الجزء الأول من البرنامج الثالث

```

}
int Y = getmaxy()/10;
settextstyle(TRIPLEX_FONT, HORIZ_DIR, 0);
int H = textwidth("A");
MyMouse.show_cursor(); أظهر المؤشر
outtextxy(0,Y,"Press a key to hide the cursor.");
getch();
MyMouse.hide_cursor(); أخف المؤشر
outtextxy(0,Y+2*H,"Press a key to restore\
the cursor.");
getch();
MyMouse.show_cursor();
outtextxy(0,Y+4*H,"Press a key to exit.");

```

```

    getch();
    closegraph();
    return(0);
}
// The initialize-mouse function:
int mouse::initialize(void)
{
    union REGS in_regs;
    union REGS out_regs;
    // Write a value in the input register:
    in_regs.x.ax = RESET;
    // Invoke the interrupt 33h:
    int86(0x33, &in_regs, &out_regs);
    // Return the value of the output register:
    return(out_regs.x.ax);
}
// The show-cursor function.
void mouse::show_cursor(void)
{
    union REGS in_regs;
    union REGS out_regs;
    // Write the value 1 into AX:
    in_regs.x.ax = SHOW;
    int86(0x33, &in_regs, &out_regs);
}
// The hide-cursor function:
void mouse::hide_cursor(void)
{
    union REGS in_regs;
    union REGS out_regs;
    // Write the value 2 into AX:
    in_regs.x.ax = HIDE;
    int86(0x33, &in_regs, &out_regs);
}

```

شكل (١٦)

الجزء الثاني والأخير من البرنامج الثالث

(٦-٨) قراءة معلومات الفأر

إن دالة الخدمة رقم 3 بالجدول تستخدم في التعرف على حالة الفأر بصفة عامة . وعند استدعاء هذه الدالة فإنها تُرجع المعلومات اللازمة مخزنة في المسجلات BX ، CX ، DX .

وهذه هي الشفرة التي يحتويها المسجل BX ، وهي شفرة رقمية يعبر كل رقم منها عن حالة أحد الأزرار :

المعنى	محتويات المسجل BX
لا يوجد أى زر مضغوط	0
الزر الأيمن مضغوط	1
الزر الأيسر مضغوط	2
الزران مضغوطان	3

شكل (١٧)

محتويات المسجل BX

أما المسجل CX فهو يحتوى على رقم العمود الذى يوجد عنده المؤشر . ويحتوى المسجل DX على رقم الصف الذى يوجد عنده المؤشر . والأعداد المستخدمة للتعبير عن الصف والعمود تستخدم وحدة البكسل .

وفي هذه الفقرة سوف ننشئ دالة عامة تستخدم في الاتصال بالمسجلات الأربعة AX ، BX ، DX ، CX فتكتب فيها (كمسجلات للدخل) وتقرأ محتوياتها (كمسجلات للخروج) وسوف نطلق على هذه الدالة اسم دالة المعلومات (information) ..

وهذا هو نص الدالة :

```
void mouse::information(int *r1, int *r2,
                        int *r3, int *r4)
{
    union REGS in_regs;
    union REGS out_regs;
    in_regs.x.ax = *r1;
    in_regs.x.bx = *r2;
    in_regs.x.cx = *r3;
    in_regs.x.dx = *r4;
    int86(0x33, &in_regs, &out_regs);
    *r1 = out_regs.x.ax;
    *r2 = out_regs.x.bx;
    *r3 = out_regs.x.cx;
    *r4 = out_regs.x.dx;
}
```

الكَلَابَة في السجَلات

قراءة السجَلات

شكل (١٨)

وكما نرى من إعلان الدالة أنها تستخدم المؤشرات كبارامترات حتى يمكن استخدام الدالة في تغيير محتويات المسجلات إذا اقتضى الأمر . ولأننا قد عرفنا الآن وظيفة المسجلات جميعاً فيمكننا توظيف هذه الدالة كما نشاء .

فالبارامتر الأول r1 مثلاً يمثل رقم دالة الخدمة التي سوف تكتب في المسجل AX . أما البارامترات الثلاثة الأخرى فتمثل المسجلات BX ، CX ، DX بالترتيب .

فاذا أردنا قراءة إحداثيات المؤشر على الشاشة فكل ما علينا أن نمنح البارامتر r1 القيمة العددية 3 (وهي قيمة دالة الخدمة رقم 3) ثم نقرأ القيمة المرتجعة للبارامترات r3 ، r4 .

وفي الشكل التالي قد وضعنا علامة التعليق أمام العبارات الزائدة عن الحاجة (في حالة قراءة إحداثيات المؤشر) ويجوز الإبقاء عليها فهي لا تأثير لها على النتيجة في هذا التطبيق .

```
void mouse::information(int *r1, int *r2,
                       int *r3, int *r4)
{
    union REGS in_regs;
    union REGS out_regs;
    in_regs.x.ax = *r1;
    // in_regs.x.bx = *r2;
    // in_regs.x.cx = *r3;
    // in_regs.x.dx = *r4;
    int86(0x33, &in_regs, &out_regs);
    *r1 = out_regs.x.ax;
    *r2 = out_regs.x.bx;
    *r3 = out_regs.x.cx;
    *r4 = out_regs.x.dx;
}
```

شكل (١٩)

ولكى نستخدم هذه الدالة الجديدة فى قراءة الإحداثيات سوف نعرف ثابتاً جديداً من ثوابت الفأر وهو :

```
const GET-STATUS=3
```

وبتخصيص هذا الثابت للبارامتر الأول للدالة فإنه يمكن قراءة الإحداثيات من البارامتريين الثالث والرابع مباشرة كما فى شريحة البرنامج الآتية :

```
int r1 = GET_STATUS, r2, x, y;
MyMouse.information(&r1,&r2,&x,&y);
sprintf(String, "Cursor position now is (%d,%d)",x,y);
outtextxy(0,Y+6*H,String);
```

شكل (٢٠)

والشكل الآتى يوضح مثلاً لتنفيذ البرنامج بعد إضافة الدالة الجديدة إليه حيث نرى المؤشر على الشاشة فى الموقع (441,249) ونرى التقرير المكتوب بالبرنامج .

ومن الجدير بالذكر أنك لو قمت بتشغيل البرنامج بدون تحريك الفأر على الإطلاق فإنك تحصل على إحداثيات منتصف الشاشة (أى "320,240" فى

حالة الشاشة (VGA) ، أما لو أنك قد حركت الفأر إلى موقع مختلف فسوف تحصل على الإحداثيات الجديدة كما نرى بالشكل :

Press a key to hide the cursor.
Press a key to restore the cursor.
Cursor position now is (73, 220)
Press a key to exit.

شكل (٢١)

وفيما يلي نص البرنامج الرابع كاملاً .

```
/* Program 8-4.cpp */
// Get cursor information
#include <dos.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <graphics.h>
#include <iostream.h>
// The mouse class declaration:
// to go into the header file "mouse.h"
class mouse {
public:
```

```

virtual int initialize(void);
virtual void show_cursor(void);
virtual void hide_cursor(void);
virtual void information(int *r1,
                        int *r2,
                        int *r3,
                        int *r4);
};
// The mouse consatants:
// to go into the header file "mouscnst.h"
const int RESET = 0;
const int SHOW = 1;
const int HIDE = 2;
const int GET_STATUS = 3; ثابت التعرف على الحالة
main()
{
    int driver = DETECT, mode, errorcode;
    char String[80];
    mouse MyMouse; // The mouse object
    initgraph(&driver, &mode, "d:\\tc\\bgi");
    // Graphics initialization:
    errorcode = graphresult();
    if (errorcode != grOk) {
        cout << "Graphics error:"
             << grapherrormsg(errorcode) << endl
             << "Make sure the path to graphics "
             << "drivers is correct." << endl
             << "Press any key to exit";
        getch();
        clrscr(); exit(1);
    }
}

```

شكل (٢٢)

الجزء الأول من البرنامج الرابع

```

// Mouse initialization:
if (!MyMouse.initialize()) {
    cout << "Mouse could not be initialized."
         << endl << "Press any key to exit";
    getch();
    clrscr(); exit(1);
}
//

```

```

    int Y = getmaxy()/10;
    settextstyle(TRIPLEX_FONT, HORIZ_DIR, 0);
    int H = textwidth("A");
//
    MyMouse.show_cursor();
    outtextxy(0,Y,"Press a key to hide the cursor.");
    getch();
//
    MyMouse.hide_cursor();
    outtextxy(0,Y+2*H,"Press a key to restore\
the cursor.");
    getch();
    MyMouse.show_cursor();
// Get cursor info:
    int r1 = GET_STATUS, r2, x, y;
    MyMouse.information(&r1,&r2,&x,&y);
    sprintf(String,
        "Cursor position now is (%d,%d)",x,y);
    outtextxy(0,Y+4*H,String);
    outtextxy(0,Y+6*H,"Press a key to exit.");
    getch();
    closegraph();
    return(0);
}
// The initialize-mouse function:
int mouse::initialize(void)
{
    union REGS in_regs;
    union REGS out_regs;
// Write a value in the input register:
    in_regs.x.ax = RESET;
// Invoke the interrupt 33h:
    int86(0x33, &in_regs, &out_regs);
// Return the value of the output register:
    return(out_regs.x.ax);
}

```

شكل (٢٣)
الجزء الثاني من البرنامج الرابع

```
// The show-cursor function.
void mouse::show_cursor(void)
{
    union REGS in_regs;
    union REGS out_regs;
    // Write the value 1 into AX:
    in_regs.x.ax = SHOW;
    int86(0x33, &in_regs, &out_regs);
}
// The hide-cursor function:
void mouse::hide_cursor(void)
{
    union REGS in_regs;
    union REGS out_regs;
    // Write the value 2 into AX:
    in_regs.x.ax = HIDE;
    int86(0x33, &in_regs, &out_regs);
}
void mouse::information(int *r1, int *r2,
                        int *r3, int *r4)
{
    union REGS in_regs;
    union REGS out_regs;
    in_regs.x.ax = *r1;
    in_regs.x.bx = *r2;
    in_regs.x.cx = *r3;
    in_regs.x.dx = *r4;
    int86(0x33, &in_regs, &out_regs);
    *r1 = out_regs.x.ax;
    *r2 = out_regs.x.bx;
    *r3 = out_regs.x.cx;
    *r4 = out_regs.x.dx;
}
```

شكل (٢٤)

الجزء الثالث والأخير من البرنامج الرابع

ونلاحظ في هذا البرنامج أننا قد حذفنا الرسالة الخاصة بنجاح إعداد الفأر واكتفينا بالرسالة التي تظهر في حالة الفشل وذلك حتى لا تزدحم شاشة البرنامج التطبيقي بما هو خارج عن حدود التطبيق .

(٨ - ٧) تنظيم البرنامج

قد أنشأنا حتى الآن مجموعة لا بأس بها من الدوال بحيث يقتضى الأمر أن نقف وقفة للمراجعة والتنظيم . وهذا ملخص بالخطوات التنظيمية التي سنجرىها على البرنامج قبل أن نشرع فى تطوير جديد .

[١] وضع إعلان الفصيلة "mouse" فى ملف مستقل بالاسم :

mouse.h

[٢] وضع ثوابت الفأر فى ملف مستقل بالاسم :

mouscnst.h

[٣] وضع جميع دوال الفأر فى ملف مستقل بالاسم :

mousefns.cpp

[٤] يبقى بعد ذلك دالة البرنامج الرئيسى ولنطلق عليها الاسم المسلسل للبرامج وليكن .

8-5.cpp

وسوف يحتوى البرنامج الرئيسى عل عبارات تضمين ملفات العناوين السابقة
أى :

```
#include "mouse.h"
#include "mouscnst.h"
```

كما يجب تضمين هذه الملفات أيضاً فى ملف الدوال رقم (3) .

[٥] ينضم الملفان رقم ٣ ورقم ٤ لتكوين مشروع واحد نمثله الاسم :

8-5.prj

وسوف نتبع هذا النظام في سائر المشروعات القادمة .

وفيما يلي نستعرض الوحدات الجديدة للبرنامج :

© أولاً - ملف **الفصيلة mouse.h** :

```
// Mouse class declarations: "mouse.h"
#ifndef MOUSE_H
#define MOUSE_H
class mouse {
private:
    union REGS in_regs;
    union REGS out_regs;
public:
    ← أعضاء عامة
    virtual int initialize(void);
    virtual void show_cursor(void);
    virtual void hide_cursor(void);
    virtual void information(int *r1,
                             int *r2,
                             int *r3,
                             int *r4);
};
#endif ← نهاية الماكرو الشرطى
```

شكل (٢٥)

وأول ما نلاحظه على هذا الملف هو استخدام الماكرو الشرطى :

```
#ifndef
.....
.....
#endif
```

وهذا هو المتبع عندما تتعدد ملفات العناوين ويتكرر إعلانها في أكثر من ملف (راجع مبادئ لغة سى++ للمؤلف) .

أما الملاحظة الهامة هنا فهي إعلان المنشآت المشتركة "in-regs" ،
 "out-regs" كأعضاء خاصة للفصيلة ؛ وهذا يغني عن تكرارها في سائر
 الدوال الأعضاء . ويجوز إعلانها باستخدام كلمة protected إذا كان في نيتك
 توريث الفصيلة لفصائل أخرى .

© ثانياً - ملف الثوابت "mouscnst.h" :

```
// Mouse constants: "mouscnst.h"
#ifndef MOUSCNST_H
#define MOUSCNST_H
const int RESET = 0;
const int SHOW = 1;
const int HIDE = 2;
const int GET_STATUS = 3;
#endif
```

شكل (٢٦)

ولا جديد هنا سوى استخدام الماكرو الشرطي "#ifndef" ، ومن
 الجدير بالذكر أن هذا الماكرو يستخدم عادة من باب الاحتياط تجنباً لتكرار
 الإعلانات ؛ وهي عادة طيبة أن تستخدمه دائماً في ملفات العناوين .

© ثالثاً - ملف دوال الفأر "mousefns.cpp" :

أما هذا الملف فهو أحد ملفات المشروع وهو يحتوي على جميع دوال
 الفأر بعد حذف إعلانات المنشآت المشتركة منها ، وإضافة ملفات العناوين
 للثوابت وللـفصيلة علاوة على ملف العناوين "dos.h" الذي يحتوي على
 إعلان منشآت المسجلات .



```
// The mouse functions file "mousefns.cpp"
#include <dos.h>
#include "mouse.h"
#include "mouscnst.h"
//
// The initialize-mouse function:
int mouse::initialize(void)
{
// Write the value 0 into AX:
    in_regs.x.ax = RESET;
    int86(0x33, &in_regs, &out_regs);
    return(out_regs.x.ax);
}
// The show-cursor function:
void mouse::show_cursor(void)
{
// Write the value 1 into AX:
    in_regs.x.ax = SHOW;
    int86(0x33, &in_regs, &out_regs);
}
// The hide-cursor function:
void mouse::hide_cursor(void)
{
// Write the value 2 into AX:
    in_regs.x.ax = HIDE;
    int86(0x33, &in_regs, &out_regs);
}
// The communication function:
void mouse::information(int *r1, int *r2,
                        int *r3, int *r4)
{
    in_regs.x.ax = *r1;
    in_regs.x.bx = *r2;
    in_regs.x.cx = *r3;
    in_regs.x.dx = *r4;
    int86(0x33, &in_regs, &out_regs);
    *r1 = out_regs.x.ax;
    *r2 = out_regs.x.bx;
    *r3 = out_regs.x.cx;
    *r4 = out_regs.x.dx;
}
}
```

شكل (٢٧)

© رابعاً - ملف الدالة الرئيسية (البرنامج الرئيسى) .

أما هذا الملف فهو يحمل الاسم "8-5.cpp" لأنه يتغير دائماً بحسب منطق البرنامج المطلوب وهذا هو الملف الأساسى الذى يعمل فيه المبرمج . أما ملفات الدوال والعناوين (بعد أن تكتمل معالمها) فإنها تصبح جزءاً ثابتاً من مكتبة أدوات الفأر يمكنك استخدامها فى أى تطبيق .

والبرنامج "8-5.cpp" الذى هو صلب المشروع "8-5.prj" يودى إلى نفس النتيجة كما البرنامج السابق (8-4.cpp) ولكنه مكتوب بصورة مختلفة ويدخل ضمن مكونات المشروع .



```

/* Program 8-5.cpp */
// Get cursor information
#include <dos.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <graphics.h>
#include <iostream.h>
#include "mouse.h"
#include "mouscnst.h"
//
main()
{
    int driver = DETECT, mode, errorcode;
    char String[80];
    mouse MyMouse; // The mouse object
    initgraph(&driver, &mode, "d:\\tc\\bgi");
    // Graphics initialization:
    errorcode = graphresult();
    if (errorcode != grOk) {
        cout << "Graphics error:"
             << grapherrormsg(errorcode) << endl
             << "Make sure the path to graphics "
             << "drivers is correct." << endl
             << "Press any key to exit";
        getch();
        clrscr(); exit(1);
    }
    // Mouse initialization:
    if (!MyMouse.initialize()) {
        cout << "Mouse could not be initialized."
             << endl << "Press any key to exit";
        getch();
        clrscr(); exit(1);
    }
    //
    int Y = getmaxy()/10;
    settextstyle(TRIPLEX_FONT, HORIZ_DIR, 0);
    int H = textwidth("A");
    //
    MyMouse.show_cursor();
    outtextxy(0, Y, "Press a key to hide the cursor.");
    getch();
}

```

شكل (٢٨)

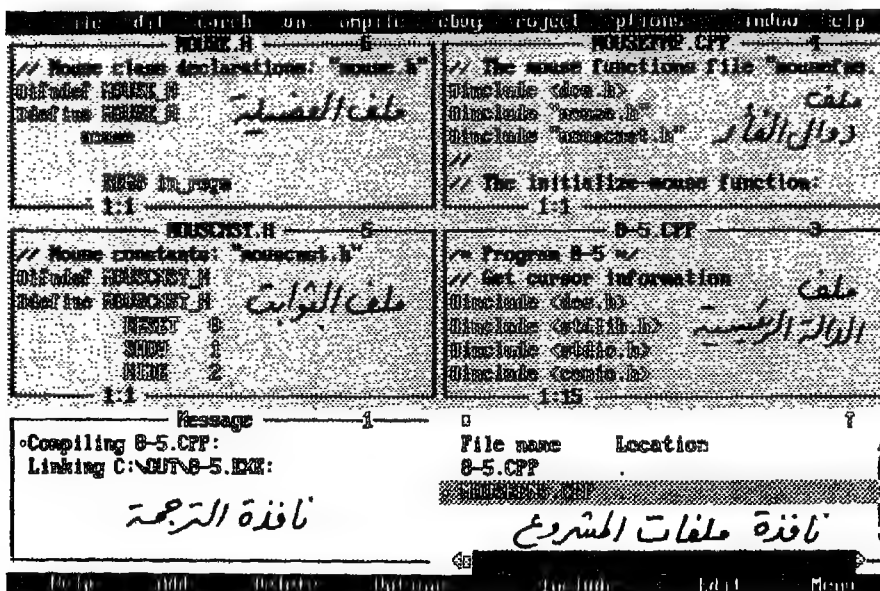
الجزء الأول من الملف "8-5.cpp"

```
//
    MyMouse.hide_cursor();
    outtextxy(0,Y+2*H,"Press a key to restore\
the cursor.");
    getch();
    MyMouse.show_cursor();
// Get cursor info:
    int r1 = GET_STATUS, r2, x, y;
    MyMouse.information(&r1,&r2,&x,&y);
    sprintf(String,
        "Cursor position now is (%d,%d)",x,y);
    outtextxy(0,Y+4*H,String);
    outtextxy(0,Y+6*H,"Press a key to exit.");
    getch();
    closegraph();
    return(0);
}
```

شكل (٢٩)

الجزء الثاني والأخير من الدالة الرئيسية للمشروع الخامس

أما الشكل التالي فهو يوضح شاشة المترجم بورلاند سي++ وعليها ملفا المشروع علاوة على ملفات العناوين مفتوحة على شاشة واحدة :



شكل (٣٠)

تدريب (٨ - ١)

© دالة تحريك المؤشر :

انظر إلى تنفيذ البرنامج بالشكل التالي . لقد أضفنا خطوة جديدة إلى البرنامج تجعل المؤشر ينتقل من مكانه الحالي بمنتصف الشاشة إلى مكان جديد محدد بإحداثي معين (النقطة 1,1) .

إن الدالة رقم 4 من دوال الخدمة تؤدي هذه المهمة . فلو أنك كتبت الرقم 4 في المسجل AX والإحداثيات المطلوبة في المسجلات CX ، DX فإن هذا يؤدي إلى تحريك المؤشر .

أنشئ الدالة العضوة الجديدة move—cursor وضمها إلى فصيلة الفأر . سوف تجد حل التدريب على القرص تحت اسم المشروع DRL8-1.PRJ ، أو في الفقرات القادمة .

Press a key to hide the cursor.
Press a key to restore the cursor.
Cursor position now is (320, 240)
Press a key to put the cursor in (1, 1)
Press a key to exit.

شكل (٣١)

٨-٨ (الاستجابة لأزرار الفأر)

حتى الآن فإننا نعتمد على أزرار اللوحة لكي تستجيب لتعليمات البرنامج "Press any key..". أما الفأر فرغم أنه حر الحركة على الشاشة لكنه لا يتحكم في المدخلات . وفي هذه الفقرة سوف ننشئ الدوال التي بها يشعر البرنامج بأننا قد ضغطنا على أحد زرى الفأر ويستجيب لذلك .

ولنبدأ بإنشاء دالة منطقية تشعر بالضغط على الأزرار وترجع القيمة TRUE عند الضغط على أحد أزرار الفأر .

فلنبدأ أولاً بإضافة بعض الثوابت إلى ملف الثوابت "mousecst.h" :

```
const int FALSE=0;
const int TRUE=1;
const int LEFT=0;
const int RIGHT=1;
const int PRESSED=5;
```

شكل (٣٢)

وكما نرى فى الشكل فإن الثابت PRESSED يحمل القيمة 5 وهو رقم دالة خدمة الفأر التي تحسب عدد مرات ضغط الزر الأيسر .

أما الثوابت LEFT ، RIGHT فهي تمثل الأرقام المستخدمة للدلالة على الزر الأيسر والأيمن عند استدعاء دالة المعلومات للكتابة فى المسجل BX أى :

- الزر الأيسر الرقم 0
- الزر الأيمن الرقم 1
- الزر الأيسر أو الأيمن الرقم 2

أضف أيضاً إلى ملف الدوال mousefns.cpp الدالة الآتية :

```
// Test a pressed button:
int mouse::CheckIfPressed(int button)
{
    int m1,m2,m3,m4;
    m1=PRESSED;
    if (button==LEFT) {
        m2=LEFT;
        information(&m1,&m2,&m3,&m4);
    }
    if (m2)
        return(TRUE);
    if(button==RIGHT) {
        m2=RIGHT;
        information(&m1,&m2,&m3,&m4);
    }
    if (m2)
        return(TRUE);
    }
    return(FALSE);
}
```

شكل (٣٣)

ولا تنس - بالطبع - أن تضيف عينة هذه الدالة إلى دوال الفصيلة mouse الموجودة بالملف "mouse.h".

والدالة كما نرى تستخدم البارامتر button الذى قد يحمل الرقم 0 (للزر الأيسر LEFT) أو الرقم 1 (للزر الأيمن RIGHT).

ويعم تمرير هذا البارامتر إلى المسجل BX باستخدام البارامتر m2. كما يتم تمرير رقم دالة الخدمة 5 (الثابت PRESSED) إلى المسجل AX. وبرجوع دالة المعلومات (information) فإن البارامتر m2 سوف يحمل القيمة الدالة على ضغط أحد الأزرار من عدمه، وتكون هذه القيمة صفراً في حالة عدم الضغط على أى زر أو أحد الأرقام 1، 2، 3 في حالة الضغط على الزر الأيمن أو الأيسر أو كليهما.

ولتجربة هذه الدالة دعنا نكتب دالة رئيسية جديدة تستخدم هذه الدالة فقط كما بالشكل التالى (الملف 8-6.cpp كجزء من المشروع 6-8).


```

/* Program 8-6.cpp */
// Check if a button is pressed
#include <dos.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <graphics.h>
#include <iostream.h>
#include "mouse.h"
#include "mouscnst.h"
//
main()
{
    int driver = DETECT, mode, errorcode;
    mouse MyMouse;          // The mouse object
    initgraph(&driver, &mode, "d:\\tc\\bgi");
    // Graphics initialization:
    errorcode = graphresult();
    if (errorcode != grOk) {
        cout << "Graphics error:"
             << grapherrormsg(errorcode) << endl
             << "Make sure the path to graphics "
             << "drivers is correct." << endl
             << "Press any key to exit";
        getch();
        clrscr(); exit(1);
    }
    // Mouse initialization:
    if (!MyMouse.initialize()) {
        cout << "Mouse could not be initialized."
             << endl << "Press any key to exit";
        getch();
        clrscr(); exit(1);
    }
    // Program code starts here:
    while (!MyMouse.CheckIfPressed(LEFT))
        cout << "Press Left Button...\n";
    while (!MyMouse.CheckIfPressed(RIGHT))
        cout << "Now Press The Right Button.\n";
    closegraph();
    return(0);
}

```

شکل (۳۴)

وعند تنفيذ هذا المشروع فإن الرسالة الآتية :

Press Left Button...

تتابع في سطور متتالية على الشاشة ، فإذا ضغطت على الزر الأيسر للفأر فإن رسالة جديدة تظهر على الشاشة بصورة متتابعة أيضاً :

Now Press The Right Button

فإذا ضغطت على الزر الأيمن انتهى البرنامج .

تدريب (٨-٢)

اكتب دالة جديدة على غرار الدالة :

CheckIfPressed

بحيث تستخدم في اختبار إطلاق أى زر من أزرار الفأر يمكنك استخدام دالة الخدمة رقم 6 في إنشاء هذه الدالة ولنطلق عليها الاسم :

CheckIfReleased

وسوف تحتاج أيضاً إلى إعلان ثابت جديد مثل :

const int RELEASED=6

وفيما يلي نص البرنامج "7-8.cpp" الذى يستخدم هذه الدالة حيث نرى على الشاشة رسالة تطلب منا استمرار الضغط على الزر الأيسر :

Keep Pressing The Left Button

فإذا ضغطت على الزر الأيسر تغير محتوى الرسالة إلى :

Now Release The Button

وبمجرد إطلاق الزر ينتهى البرنامج . (هذا هو المشروع 8-7.PRJ)

```

/* Program 8-7.cpp */
// Check if a button is released
#include <dos.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <graphics.h>
#include <iostream.h>
#include "mouse.h"
#include "mouscnst.h"
//
main()
{
    int driver = DETECT, mode, errorcode;
    mouse MyMouse;          // The mouse object
    initgraph(&driver, &mode, "d:\\tc\\bgi");
    // Graphics initialization:
    errorcode = graphresult();
    if (errorcode != grOk) {
        cout << "Graphics error:"
             << grapherrormsg(errorcode) << endl
             << "Make sure the path to graphics "
             << "drivers is correct." << endl
             << "Press any key to exit";
        getch();
        clrscr(); exit(1);
    }
    // Mouse initialization:
    if (!MyMouse.initialize()) {
        cout << "Mouse could not be initialized."
             << endl << "Press any key to exit";
        getch();
        clrscr(); exit(1);
    }
    // Program code starts here:
    while (!MyMouse.CheckIfPressed(LEFT))
        cout << "Keep Pressing The Left Button...\n";
    while (!MyMouse.CheckIfReleased(LEFT))
        cout << "Now Release the Button.\n";
    closegraph();
    return(0);
}

```

شکل (۳۵)

(٨-٩) حزمة أدوات الفأر

بهذه المجموعة من الدوال فقد اكتملت لدينا حزمة من أدوات الفأر لا بأس بها ، حيث يمكننا توظيفها في أكثر من تطبيق . كما يمكنك إذا شئت أن تضيف إلى فصيلة الفأر ما تشاء من دوال جديدة بالاستعانة بجدول دوال الخدمة للفأر .

وكما رأينا في المشروعات السابقة فإن دالة البرنامج الرئيسى ، وهى تمثيل التطبيق الحالى ، تتغير دائماً من تطبيق إلى آخر ، أما ملف الدوال الأعضاء ، بالرغم من أنه دائم النمو ، لكنه يستخدم مع أى دالة رئيسية . فعندما بدأنا هذا الباب كانت عائلة الدوال تحتوى على عدد قليل من الأعضاء حتى وصلت إلى ما هى عليه الآن ، ومع ذلك فالملف هو هو ويجوز استخدامه مع جميع الأمثلة .

وفيما يلى نعرض نصوص الملفات فى صورتها النهائية مع دالة رئيسية تمثل تطبيقاً شاملاً للاستفادة من الدوال جميعاً .

⊙ ملف الثوابت "mouscnst.h" :

هذه هى الصورة النهائية لملف الثوابت ونلاحظ إضافة ثابت جديد وهو :

const int EITHER=2

وذلك لتمثيل أى من الأزرار (الأيسر أو الأيمن) ، ولكن هذا الثابت لم يستخدم بعد ، ويجوز استخدامه فى الدالة CheckIFPressed فى عبارة مثل :

```
if (button==LEFT || button==EITHER)
```

```
// Mouse constants: "mouse.h"
#ifndef MOUSCNST_H
#define MOUSCNST_H
const int FALSE=0;
const int TRUE=1;
const int LEFT=0;
const int RIGHT=1;
const int EITHER=2;
const int RESET = 0;
const int SHOW = 1;
const int HIDE = 2;
const int GET_STATUS = 3;
const int MOVE = 4;
const int PRESSED=5;
const int RELEASED=6;
#endif
```

شكل (٣٦)

© ملف إعلان فصيلة الفأر "mouse.h" :

فيما يلي نص إعلان الفصيلة بما تحتويه من أعضاء محمية وعامة . وقد أضفنا دالة جديدة هي الدالة "MouseInput" لاستخدامها في عمليات الدخل بصفة عامة وسيلي شرحها .

```
// Mouse class declarations: "mouse.h"
#ifndef MOUSE_H
#define MOUSE_H
class mouse {
private:
    union REGS in_regs;
    union REGS out_regs;
public:
    virtual int initialize(void);
    virtual void show_cursor(void);
    virtual void hide_cursor(void);
    virtual void information(int *r1,
                           int *r2,
                           int *r3,
                           int *r4);
```

```

        virtual void move_cursor(int x,int y);
        virtual CheckIfPressed(int button);
        virtual CheckIfReleased(int button);
        int MouseInput(int button);
};
#endif

```

شكل (٣٧)

② ملف تطبيق الفصيلة "mousefns.h" :

وهذا هو ملف تطبيق الفصيلة الذى يحتوى على تعريف الدوال الأعضاء ، ونرى فيه جميع الدوال التى أنشئت حتى الآن بما فى ذلك التمرينات .

```

// The mouse functions file "mousefns.cpp"
#include <dos.h>
#include "mouse.h"
#include "mouscnst.h"
//
// The initialize-mouse function:
int mouse::initialize(void)
{
// Write the value 0 into AX:
in_regs.x.ax = RESET;
int86(0x33, &in_regs, &out_regs);
return(out_regs.x.ax);
}
// The show-cursor function:
void mouse::show_cursor(void)
{
// Write the value 1 into AX:
in_regs.x.ax = SHOW;
int86(0x33, &in_regs, &out_regs);
}
// The hide-cursor function:
void mouse::hide_cursor(void)
{
// Write the value 2 into AX:

```

```

        in_regs.x.ax = HIDE;
        int86(0x33, &in_regs, &out_regs);
    }
    // The communication function:
    void mouse::information(int *r1, int *r2,
                           int *r3, int *r4)
    {
        in_regs.x.ax = *r1;
        in_regs.x.bx = *r2;
        in_regs.x.cx = *r3;
        in_regs.x.dx = *r4;
        int86(0x33, &in_regs, &out_regs);
        *r1 = out_regs.x.ax;
        *r2 = out_regs.x.bx;
        *r3 = out_regs.x.cx;
        *r4 = out_regs.x.dx;
    }
    // The move-cursor function:
    void mouse::move_cursor(int x,int y)
    {
        int r1 = MOVE, r2;
        information(&r1,&r2,&x,&y);
    }

```

شكل (٣٨)

الجزء الأول من ملف تطبيق الفصيلة

```

// Test a pressed button:
int mouse::CheckIfPressed(int button)
{
    int m1, m2, m3, m4;
    m1 = PRESSED;
    if (button == LEFT) {
        m2 = LEFT;
        information(&m1,&m2,&m3,&m4);
    }
    if (m2)
        return(TRUE);
    if(button == RIGHT) {
        m2 = RIGHT;
    }
}

```

```

        information(&m1, &m2, &m3, &m4);
    if (m2)
        return(TRUE);
    }
    return(FALSE);
}
// Test a released button:
int mouse::CheckIfReleased(int button)
{
    int m1, m2, m3, m4;
    m1 = RELEASED;
    if (button == LEFT) {
        m2 = LEFT;
        information(&m1, &m2, &m3, &m4);
        if (m2)
            return(TRUE);
    }
    if (button == RIGHT) {
        m2 = RIGHT;
        information(&m1, &m2, &m3, &m4);
        if (m2)
            return(TRUE);
    }
    return(FALSE);
}
// Mouse input function:
int mouse::MouseInput(int button)
{
    if (CheckIfPressed(button)) {
        // Do nothing until the button is released
        while (!CheckIfReleased(button));
        return(TRUE);
    }
    return(FALSE);
}

```

شكل (٣٩)

الجزء الثانى والأخير من ملف تطبيق الفصيلة

أما الدالة الجديدة MouseInput فهي تستخدم كلاً من دالة الضغط على الزر "CheckIfPressed" ودالة إطلاق الزر "CheckIfReleased" لاستقبال ضغطة زر من الفأر. فنحن نعلم أن الضغطة الواحدة (وهي تسمى click)

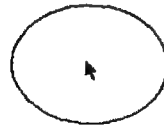
تتكون من عمليتين : الضغط والإطلاق . ولذلك فإن منطق هذه الدالة يجعل البرنامج يشعر بالضغط على الزر ثم ينتظر حتى يُطلق الزر ثم تُرجع الدالة القيمة TRUE . والحلقة التكرارية المستخدمة في الانتظار هي الحلقة while ونلاحظ وجود فاصلة منقوطة في نهايتها .

© ملف الدالة الرئيسية (تطبيق) :

فيما يلي نقدم برنامجاً تطبيقياً يجمع هذه الأدوات في تطبيق واحد . وعند تشغيل هذا البرنامج يظهر شكل بيضاوى أسفل الشاشة وتتابع على الشاشة مجموعة من الرسائل كالموضحة بالشكل التالي وهي تؤدي إلى إظهار وإخفاء المؤشر وقراءة إحداثياته وتحريكه إلى مركز الشكل البيضاوى ويتم جميع المُدخلات باستخدام الزر الأيسر للفأر (كما هو المعتاد في البرامج) ويتم إنهاء البرنامج باستخدام الزر الأيمن .

Click left button to hide the cursor.
Click left button to restore the cursor.
Cursor position now is (320, 240)
Click left button to put the cursor
at the ellipse center.
Click the right button to exit.

تنفيذ المشروع الثامن



شكل (٤٠)

تنفيذ البرنامج

والدالة الرئيسية الموضحة بعد هي جزء من المشروع الثامن الذي يحمل
الاسم 8-8.PRJ وهو يتكون من :

١ — الملف mousefn.cpp

٢ — الملف 8-8.cpp

```
/* Program 8-8.cpp */
// General Application
#include <dos.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <graphics.h>
#include <iostream.h>
#include "mouse.h"
#include "mouscnst.h"
const int SIZE = 4;
//
main()
{
    int driver = DETECT, mode, errorcode;
    char String[80];
    mouse MyMouse;          // The mouse object
    initgraph(&driver, &mode, "d:\\tc\\bgi");
// Graphics initialization:
    errorcode = graphresult();
    if (errorcode != grOk) {
        cout << "Graphics error:"
              << grapherrormsg(errorcode) << endl
              << "Make sure the path to graphics "
              << "drivers is correct." << endl
              << "Press any key to exit";
        getch();
        clrscr(); exit(1);
    }
// Mouse initialization:
    if (!MyMouse.initialize()) {
        cout << "Mouse could not be initialized."
              << endl << "Press any key to exit.";
        getch();
        clrscr(); exit(1);
    }
//
```

الدالة الرئيسية
للمشروع الثامن
الجزء الأول

```

int X = getmaxx()/10;
int Y = getmaxy()/10;
settextstyle(SANS_SERIF_FONT, HORIZ_DIR, SIZE);
ellipse(9*X,9*Y,0,360,X,Y);
int H = 2*textwidth("A");
//

```

شكل (٤١)

الجزء الأول من الملف "8-8.cpp"

الدالة الرئيسية للمشروع الثامن
الجزء الثاني

```

MyMouse.show_cursor();
outtextxy(0,Y,"Click left button to hide\
the cursor.");
while (!MyMouse.MouseInput(LEFT));
MyMouse.hide_cursor();
outtextxy(0,Y+H,"Click left button to\
restore the cursor.");
while (!MyMouse.MouseInput(LEFT));
MyMouse.show_cursor();
// Get cursor info:
int r1 = GET_STATUS, r2, x, y;
MyMouse.information(&r1,&r2,&x,&y);
sprintf(String,
    "Cursor position now is (%d,%d)",x,y);
outtextxy(0,Y+2*H,String);
outtextxy(0,Y+3*H,"Click left button\
to put the cursor");
outtextxy(0,Y+4*H,"at the ellipse center.");
while (!MyMouse.MouseInput(LEFT));
MyMouse.move_cursor(9*X,9*Y);
outtextxy(0,Y+5*H,"Click the right button\
to exit.");
while (!MyMouse.MouseInput(RIGHT));
closegraph();
return(0);
}

```

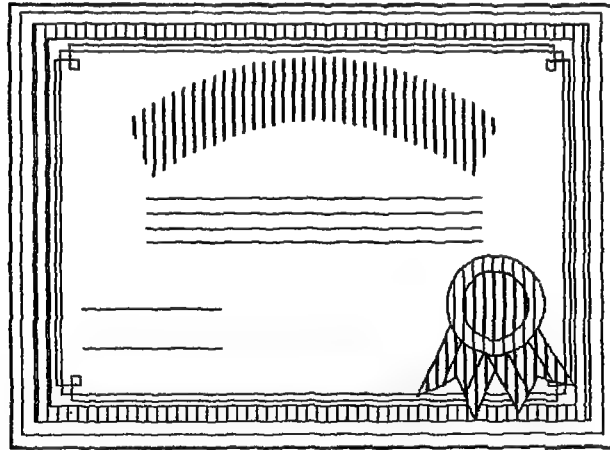
شكل (٤٢)

الجزء الثاني والأخير من الملف "8-8.cpp"

افكار للتطوير

إن البرنامج الجيد هو الذى يكون مستعداً لكل الاحتمالات فإذا كان الفأر متصلاً بالكمبيوتر قام بتشغيله كما رأينا فى هذ الباب ؛ أما إذا كان الفأر غير جاهز لسبب أو آخر فعلى البرنامج أن يستمر بما لديه من موارد . والفكرة هنا أن ننشئ فصيلة جديدة تحاكي الفأر لكنها تستخدم لوحة الأزرار فى حالة غياب الفأر . ومن المحاكاة أن نستخدم أزرار الأسهم مثلاً لتحريك مؤشر الفأر فى الاتجاهات المختلفة وأن نستخدم بعض الأزرار الخاصة مثل أزرار الدوال لمحاكاة الزر الأيمن والزر الأيسر .

والفصيلة الجديدة لن تبدأ من الصفر ، فبالوراثة تستطيع أن تقتصد وتستفيد من كل ما كتبه من دوال . والمجال مفتوح للإبداع .



شكل (٤٣)

إلى لقاء قادم مع "تيربو فيزيون"

كان هذا ختام اللقاء مع أدوات لغة سى++ للرسم . ومع ذلك فبالرغم من أننا كتبنا جميع البرامج بلغة سى++ لكننا لم نوف البرمجة الموجهة نحو الأهداف (OOP) حقها في هذا الكتاب . فالمكتبة "TVISION" عامرة بالفصائل التي كتبها مبرمجو شركة بورلاند والتي تشكل في مجموعها أدوات قوية لخلق النوافذ والقوائم في بيئة نظام التشغيل "دوس" بدون أن تحتاج إلى دخول بيئة النوافذ (Windows) .

إنها تمكنتك من خلق النوافذ بدون برنامج النوافذ . فلنتواعد على لقاء آخر حول هذا الموضوع ، ودائماً إلى اللقاء ...

والله ولي التوفيق ...

مهفيس/إسماعيل الحسيني



■ الملاحق ■

الملحق (أ)



أهم المصطلحات المعربة

أولاً :

المصطلحات المعربة فى مجال
الكومبيوتر مفهارة أبجدياً

• **ANSI (American National Standards Institute)**

هيئة القياسات الأمريكية

• **Array**

- * Multi – dimensional array مصفوفة ذات عدة أبعاد
- * One – dimensional array (متجه) مصفوفة ذات بعد واحد
- * Two – dimensional array مصفوفة ذات بعدين

• **ASCII (American Standard Code for Information Interchange)**

الكود "آسكي"

* **BIT**

بت (رقم ثنائي)

• **Buffer / Memory buffer**

وعاء في الذاكرة

* **BYTE**

- **B (Byte)** بايت
- **GB (Gigabyte)** جيجا بايت
- **KB (KiloByte)** كيلو بايت
- **MB (MegaByte)** ميغا بايت
- **TB (terabyte)** تيرا بايت

* **Character**

لبنة (حرف أو رقم أو علامة خاصة)

* **Compilation**

ترجمة (تجميعية)

* **Compiler**

المترجم (التجميعي)

* **Computer**

- **Computerized machines** الماكينات المبرمجة
- **Home computer** كومبيوتر منزلي
- **Mainframe computer** كومبيوتر كبير
- **Micro computer** ميكرو كومبيوتر
- **Mini computer** كومبيوتر متوسط (ميني كومبيوتر)
- **Personal Computer (PC)** كومبيوتر شخصي
- **Special Purpose computer** كومبيوتر ذو غرض خاص

• Super computer	كومبيوتر فائق
• Work Station	محطة عمل
* Constant	
• constant, Literal constant/Constant	ثابت
• Named constant	ثابت مُسمّى
• String / String constant	ثابت حرفي
* CPU (Central Processing Unit)	وحدة المعالجة المركزية
* Data	
• Data	بيانات
• Identifier	اسم المتغير
• Information	معلومات
• Data name / Variable	متغير/اسم بيان
• Data Processing (DP)	معالجة البيانات
• Data value	قيمة البيان
* Decision	قرار
* Digit	رقم (لاحظ أن العدد Number قد يتكون من عدة أرقام)
* Disk	
• Floppy disk/diskette	قرص مرن
• Hard disk	قرص صلب
• Sector	قطاع
• Track	مسلك
* Error message	رسالة خطأ
* Expression	تعبير
* File	
• Date File	ملف بيانات
• Executive file	ملف تنفيذي
• Field / Item	حقل

- Header file ملف عناوين
- Object file ملف هدف
- Program file ملف برنامج
- Record سجل
- source file الملف المصدر
- * Format
 - Format صيغة أو فورمات
 - Format specifiers / Conversion specifiers
مُوصفات الفورمات أو مُوصفات التحويل
- * Function
 - Function block بلوك الدالة
 - Function definition تعريف الدالة
 - Function header عنوان الدالة
 - Function prototype عينة الدالة
- * Hardcopy الخرج المطبوع
- * I/O
 - Output device جهاز الخرج
 - Input device جهاز الدخل
- * Integrated Circuit (IC) / Chip دائرة متكاملة / شريحة
- * Interpreter مترجم فوري
- * Language
 - Assembly Language لغة التجميع
 - High Level Languages (HLL) لغة عالية المستوى
 - Low Level Languages (LLL) لغة منخفضة المستوى
 - Machine Language لغة الماكينة
- * Letter حرف (أبجدي)

- * Linker برنامج الربط
- * Loop
 - Infinite Loop حلقة تكرارية لا نهائية
- * Memory
 - Auxiliary memory الذاكرة المساعدة (القرص)
 - Backing storage المخزن المساعد (القرص)
 - Live memory الذاكرة الحية (رام)
 - Main memory الذاكرة الرئيسية (رام)
 - Primary storage المخزن الأساسي (رام)
 - Internal memory الذاكرة الداخلية (رام)
 - RAM (Random Access Memory) الذاكرة رام
 - Read – write memory ذاكرة قراءة وكتابة (رام)
 - Volatile memory الذاكرة المتطايرة (رام)
- * Micro Processor المعالج الميكروى
- * Module وحدة من وحدات البرنامج
- * Multitasking تشغيل أكثر من برنامج في نفس الوقت
- * Multiuser استخدام الكمبيوتر بأكثر من شخص في نفس الوقت
- * Network
 - LAN (Local Area Network) شبكة محلية
 - WAN (Wide Area Network) شبكة واسعة
- * OOP (Object Oriented Programming) البرمجة الموجهة نحو الأهداف
- * Operator مؤثر (لا تخلط مع المؤشر pointer)
- * Parameter / Argument دليل / بارامتر
- * Peripherals
 - Disk قرص (مغناطيسى)
 - CD ROM قرص ضوئى للقراءة فقط (روم)
 - Optical disk قرص ضوئى (قرص ليزر)

• Joystick	أداة ألعاب
• Keyboard	لوحة أزرار
• Modem	جهاز اتصال بالتليفون
• Monitor	شاشة عرض
• Mouse	فأر / فأر إلكتروني
• Plotter	جهاز توقيع
• Printer	طابعة / جهاز طباعة
• Scanner	جهاز مسح
* Pointer	مؤشر
* Program	برنامج
* Programming	برمجة
* Run / Execution	تنفيذ/تشغيل (البرنامج)
* Statement	عبارة
* Structure	
• Control structure	منشأ تحكم
• Data structure	منشأ بيانات
* Subroutine	روتين فرعي
* Subprogram	برنامج فرعي
* syntax	
• syntax error	خطأ في قواعد اللغة
* Ware	
• Firmware	برنامج مكتوب على شريحة
• Hardware	المعدات
• Software	البرمجيات

ثانياً :

أهم المصطلحات المعربة
في لغتي سي ، سي ++

البيانات : DATA

- Character لبنة
- Digit رقم
- Number عدد
- Numeric character لبنة رقمية
- Alphabetic character / Letter حرف
- Character array مصفوفة لبنات
- Character pointer مؤشر حرفي / مؤشر إلى لبنة
- Character constant ثابت لبنة
- String / String constant حرفي/ثابت حرفي
- Numeric constant ثابت عددي
- Integer constant ثابت صحيح
- Float (floating point/real) constant ثابت حقيقي
- Operator مؤثر
- High Level operators المؤثرات عالية المستوى
- Literal constant / Constant ثابت
- Named constant (const) ثابت مُسمّى
- Data name / Variable متغير / اسم بيان
- Data type / Type نمط / نوع / طراز
- User defined type نمط مبتكر
- Data value قيمة البيان
- Identifier اسم (مبتكر لتسمية المتغيرات والدوال إلى آخره ..)
- Parameter / Argument بارامتر / دليل
- Pointer مؤشر
- Pointer array مصفوفة مؤشرات

- Object array مصفوفة أهداف
- Reference مرجع
- Scientific notation الطريقة العلمية / الأسية
- Class (class) فصيلة
- Enumeration (enum) نمط قائمة / قائمة
- structure (struct) منشأ
- Union منشأ مشترك

البرنامج PROGRAM

- Block بلوك
- Module وحدة (من وحدات البرنامج)
- Statement عبارة
- Expression تعبير
- Passing parameters إمرار / تمرير البارامترات
- Passing by reference التمرير بالمرجع
- Passing by value التمرير بالقيمة
- Cast / Casting الإسقاط
- Decision قرار
- Directive توجيه
- Loop حلقة تكرارية
- Macro ماكرو (أمر مُجمّع)
- Main function (main) الدالة الرئيسية / دالة البرنامج الرئيسي
- Dynamic allocation الحجز الديناميكي للذاكرة
- Dynamic array مصفوفة ديناميكية
- Global عام (متغير عام)
- Local محلي (متغير محلي)
- Internal linkage ربط داخلي

- External linkage ربط خارجي
- Project مشروع
- Scope نطاق
- Visibility رؤية
- Initialization شحن (ابتدائي للمتغيرات)

الملفات FILES

- File ملف
- Data File ملف بيانات
- Record سجل
- Field / item حقل
- Executive file ملف تنفيذي
- Header file ملف عناوين
- Object file ملف هدف
- Program file ملف برنامج
- Source file ملف مصدر
- Project file ملف مشروع

الترجمة COMPILATION

- Compiler مترجم (تجميعي)
- Linker برنامج الربط
- Library مكتبة
- Integrated development environment (IDE)

البيئة المدمجة للمترجم "تيرو سي أو تيرو سي++"

I/O

- Format فورمات / صيغة
- Output format فورمات المخرج

- Input format فورمات الدخّل
- I/O flags رايات الفورمات (للدخّل والخرج)
- Format specifiers / Conversion specifiers موصّفات التحويل
- Format manipulators أدوات الفورمات

الدوال *FUNCTIONS*

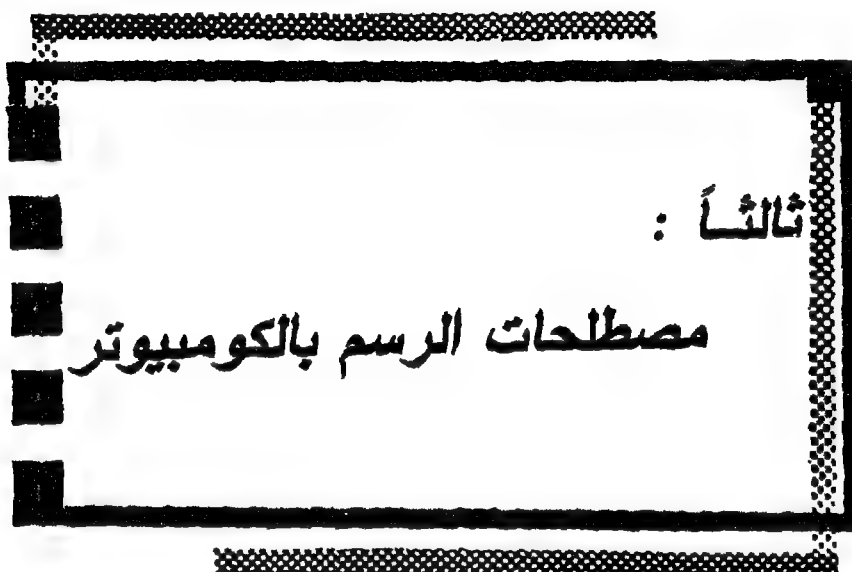
- Access functions دوال التوصل
- Conversion functions دوال التحويل
- Friend functions الدوال الأصدقاء
- Function arguments / parameters بارامترات/أدلة الدالة
- Function block بلوك الدالة
- Function definition تعريف الدالة
- Function header عنوان الدالة
- Function overloading التحميل الزائد للدالة
- Function prototye عنية الدالة
- Function return رجوع الدالة
- Function return type نوع القيمة المرتجعة من الدالة/نوع الدالة
- Function return value القيمة المرتجعة من الدالة
- Inline functions الدوال الخطيّة
- Low Level functions / routines الدوال / الروتينات منخفضة المستوى
- Member functions الدوال الأعضاء
- Operator functions دوال المؤثرات
- Parameter conversion تحويل البارامترات
- Virtual functions الدوال الافتراضية

الفصائل CLASSES

- Abstraction التجريد
- Access specifiers موصفات التوصّل
- Base class الفصيلة الأساسية / فصيلة الأساس
 - / Super class الفصيلة العليا
 - / Parent class الفصيلة الأم
- Class hierarchy شجرة العائلة للفصيلة / شجرة الوراثة
- Class composition تركيب الأهداف
- Class definition / declaration إعلان/تعريف الفصيلة
- Class implementation تطبيق الفصيلة
- Class inheritance الوراثة
- Class members أعضاء الفصيلة
- Code reusing إعادة استخدام الكود
- Constructor دالة بناء
- Destructor دالة هدم
- Data hiding / data protection حماية البيانات / إخفاء البيانات
- Data members البيانات الأعضاء
- Default constructor دوال البناء سابقة التعريف
- Derived class الفصيلة المشتقة / المستحدثة
 - / Subclass الفصيلة الدنيا
 - / Child class الفصيلة الابنة
- Dynamic object هدف ديناميكي
- Encapsulation الكبسلة
- Instance (or object) هدف

- Member functions الدوال الأعضاء
- Object (or instance) هدف
- OOP(Object Oriented Programming) البرمجة نحو الأهداف
- Polymorphism تعدد الأشكال
- Static functions الدوال الاستاتيكية
- Static member functions الدوال الاستاتيكية الأعضاء
- Operator overloading التحميل الزائد للمؤثرات
- Function overloading التحميل الزائد للدوال
- Stream objects أهداف القنوات
- Stream operators مؤثرات القنوات
- Public عام (عضو عام في الفصيلة)
- Private خاص (عضو خاص في الفصيلة)
- Protected محمي (عضو محمي في الفصيلة)
- Access level مستوى التوصل/درجة التوصل (إلى أعضاء الفصيلة)
- Access modifiers مُعدّل التوصل
- Template (class struct template) نموذج الفصيلة (تعريف الفصيلة)
- Static binding الربط الاستاتيكي للدوال
- dynamic binding الربط الديناميكي للدوال

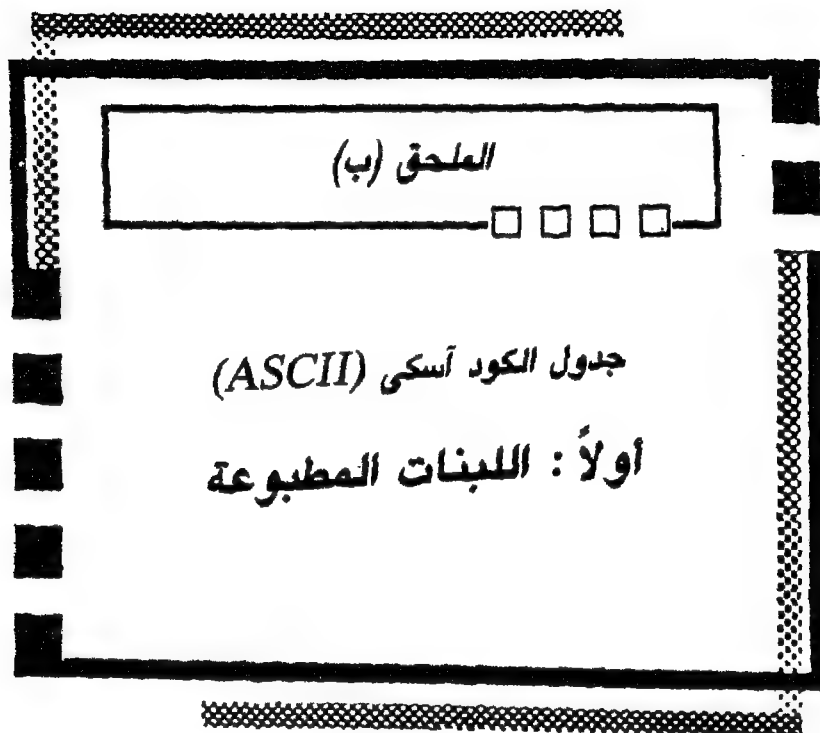




- * Video adapter موائم الرسم/كارت الرسم/كارت الفيديو
- * Graphics mode/screen نسق الرسم/بيئة الرسم
- * Textmode نسق الكتابة/بيئة النصوص
- * Graphic modes أطوار نسق الرسم
- * Video mode طور الرسم/طور الفيديو
- * Video driver ملف قيادة جهاز الفيديو
- * MDA الموائم وحيد اللون جميع هذه المصطلحات
- * CGA الموائم الملون لمجرد التعريب ولكن
- * EGA الموائم الملون المحسن المصطلحات الإنجليزية
- * VGA موائم مصفوفة الرسم المختصرة هي الشائعة
- * Hercules Adapter الموائم "هرقل" على لسان العرب .
- * Character لبنة
- * Pixel بكسلة/خلية الرسم
- * Font بنط (حجم ونوع الخط المستخدم في الكتابة)
- * Cursor مؤشر الرسم/مؤشر الفأر
- * Interrupt خط تقاطع
- * Palette لوحة الألوان
- * Pattern شبكة (الطلاء)
- * Cyan اللون التيركواز (ما بين الأزرق والأخضر)
- اللون الأرجواني (ما بين الأحمر والأزرق - خلاف اللون البنفسجي)
- * Magenta
- * Window نافذة في نسق الكتابة
- * Viewport نافذة في نسق الرسم
- * Background خلفية (الرسم أو الكتابة)
- * Clear screen / window مسح الشاشة أو النافذة
- * Fractals الفراكتلات (أسلوب جديد للرسم له أصول رياضية)

- * Fractalization عملية الفركتلة (تطبيق الفراكتلات على الرسم)
- * Filling / Painting ملء المساحات الفارغة / الطلاء
- * Rectangle مستطيل
- * Circle دائرة
- * Ellipse قطع ناقص (يجوز استخدام كلمة "بيضاوى")
- * Sector قطاع (جزء من دائرة أو من قطع ناقص)
- * Bar قضيب
- * Arc قوس
- * Poly / Polygon شكل مُضَلَّع (ذو عدة أضلاع)
- * Pie / Pieslice قطاع دائري / شريحة الفطيرة
- * Page صفحة ذاكرة (من الصفحات المخصصة للرسم)
- * Animation تحريك الأشكال على الشاشة
- * Text justification ضبط هوامش النص
- * Bar charts خرائط الأعمدة
- * Cut and Paste القص واللصق
- * Mouse الفأر (الإلكتروني)
- * Mouse buttons أزرار الفأر
- * Click ضغط زر الفأر ثم إطلاقه





THE ASCII CHARACTER SET

1. The Printable ASCII Characters

الرقم الكودي Decimal	الرقم بالنظام الثماني Octal	الرقم بالنظام السداسي عشر Hexadecimal	اللبنة Character
32	40	20	space
33	41	21	!
34	42	22	"
35	43	23	#
36	44	24	\$
37	45	25	%
38	46	26	&
39	47	27	'
40	50	28	(
41	51	29)
42	52	2a	*
43	53	2b	+
44	54	2c	,
45	55	2d	-
46	56	2e	.
47	57	2f	/
48	60	30	0
49	61	31	1
50	62	32	2
51	63	33	3
52	64	34	4
53	65	35	5
54	66	36	6
55	67	37	7
56	70	38	8
57	71	39	9
58	72	3a	:
59	73	3b	;
60	74	3c	<
61	75	3d	=
62	76	3e	>
63	77	3f	?
64	100	40	@
65	101	41	A
66	102	42	B
67	103	43	C
68	104	44	D
69	105	45	E
70	106	46	F
71	107	47	G

Decimal	Octal	Hexadecimal	Character
72	110	48	H
73	111	49	I
74	112	4a	J
75	113	4b	K
76	114	4c	L
77	115	4d	M
78	116	4e	N
79	117	4f	O
80	120	50	P
81	121	51	Q
82	122	52	R
83	123	53	S
84	124	54	T
85	125	55	U
86	126	56	V
87	127	57	W
88	130	58	X
89	131	59	Y
90	132	5a	Z
91	133	5b	[
92	134	5c	\
93	135	5d]
94	136	5e	^
95	137	5f	_
96	140	60	`
97	141	61	a
98	142	62	b
99	143	63	c
100	144	64	d
101	145	65	e
102	146	66	f
103	147	67	g
104	150	68	h
105	151	69	i
106	152	6a	j
107	153	6b	k
108	154	6c	l
109	155	6d	m
110	156	6e	n
111	157	6f	o
112	160	70	p
113	161	71	q
114	162	72	r
115	163	73	s
116	164	74	t
117	165	75	u

شكل الـ ١١٧٠

Decimal	Octal	Hexadecimal	Character
118	166	76	ص
119	167	77	و
120	170	78	خ
121	171	79	ي
122	172	7a	ز
123	173	7b	ـ
124	174	7c	ـ
125	175	7d	ـ
126	176	7e	ـ



ثانياً : لبنات الترقيم

2. The Punctuation ASCII Characters

Decimal	Octal	Hexadecimal	Character
33	41	21	!
34	42	22	"
35	43	23	#
36	44	24	\$
37	45	25	%
38	46	26	&
39	47	27	'
40	50	28	(
41	51	29)
42	52	2a	*
43	53	2b	+
44	54	2c	,
45	55	2d	-
46	56	2e	.
47	57	2f	/
58	72	3a	:
59	73	3b	;
60	74	3c	<
61	75	3d	=
62	76	3e	>
63	77	3f	?
64	100	40	@
91	133	5b	[
92	134	5c	\
93	135	5d]
94	136	5e	^
95	137	5f	_
96	140	60	`
123	173	7b	{
124	174	7c	
125	175	7d	}
126	176	7e	~

شكل اللبنة المطبوعة

ثالثاً : لبنات التحكم

3. The Control ASCII Characters

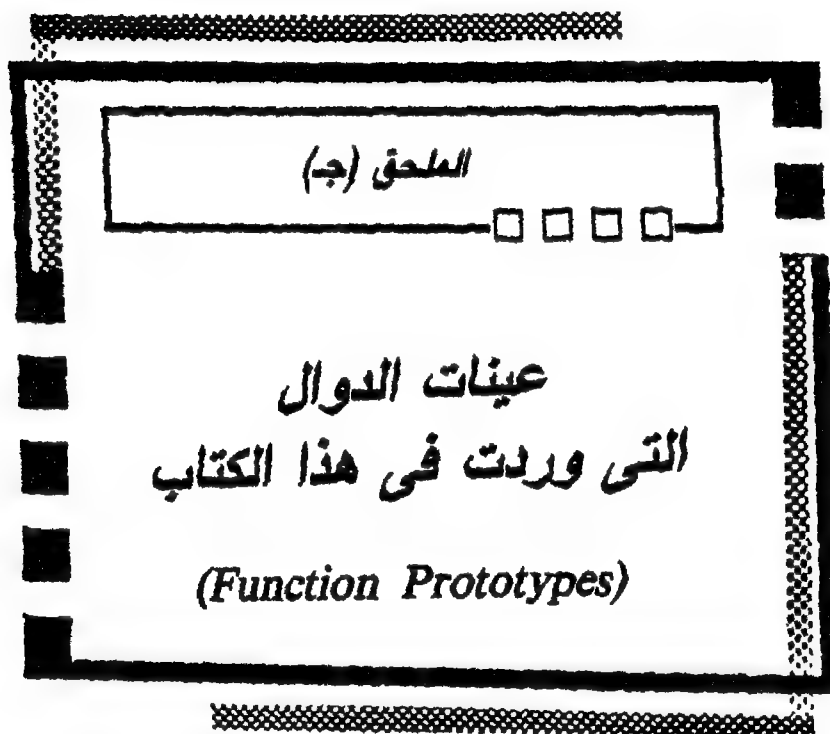
Decimal	Octal	Hexadecimal	Key	Backslash Code	Mnemonic Code
0	0	0	^@	'\0'	NUL
1	1	1	^A		SOH
2	2	2	^B		STX
3	3	3	^C		ETX
4	4	4	^D		EOT
5	5	5	^E		ENQ
6	6	6	^F		ACK
7	7	7	^G		BEL
8	10	8	^H		BS
9	11	9	^I		HT
10	12	a	^J		LF
11	13	b	^K		VT
12	14	c	^L		FF
13	15	d	^M		CR
14	16	e	^N		SO
15	17	f	^O		SI
16	20	10	^P		DLE
17	21	11	^Q		DC1
18	22	12	^R		DC2
19	23	13	^S		DC3
20	24	14	^T		DC4
21	25	15	^U		NAR
22	26	16	^V		SYN
23	27	17	^W		ETB
24	30	18	^X		CAN
25	31	19	^Y		EM
26	32	1a	^Z		SUB
27	33	1b	ESC		ESC
28	34	1c			FS
29	35	1d			GS
30	36	1e			RS
31	37	1f			US
127	177	7f	DEL		DEL

الرقم الكودي بالنظام العشري

زر التحكم المناظر

الرمز المستخدم في لغة سي

الرمز المستخدم في لغة الماكينة



```

#include <graphics.h>
void far initgraph(int far *driver,
                  int far *mode,
                  char far *path);

#include <graphics.h>
void far line(int start_X, int start_Y,
             int end_X, int end_Y);

#include <graphics.h>
void far lineto(int X, int Y);

#include <graphics.h>
void far linerel(int delta_X, int delta_Y);

#include <graphics.h>
void far circle(int x, int y,
               int radius);

#include <graphics.h>
void far setcolor(int color);

#include <graphics.h>
void far setbkcolor(int color);

#include <graphics.h>
void far detectgraph(int far driver*,
                   int far *mode);

#include <graphics.h>
int far getgraphmode(void);

#include <graphics.h>
void far setgraphmode(int mode);

```

شکل (۱)

```

#include <graphics.h>
void far getmoderange(int driver,
                      int far *lowmode,
                      int far *himode);

#include <graphics.h>
void far closegraph(void);

#include <conio.h>
void gotoxy(int x, int y);

#include <conio.h>
void textmode(int mode);

#include <conio.h>
int cprintf(const char *format [,argument,...]);

#include <conio.h>
int cputs(const char *str);

#include <conio.h>
void textcolor(int newcolor);

#include <conio.h>
void textbackground(int newcolor);

#include <conio.h>
void textattr(int newattr);

#include <conio.h>
void highvideo(void);
void lowvideo(void);
void normvideo(void);

```

شكل (٢)

```

#include <conio.h>
int getche(void);

#include <conio.h>
int putch(int ch);

#include <conio.h>
void clrscr(void);

#include <conio.h>
void window(int left, int top,
            int right, int bottom);

#include <conio.h>
char *cgets(char *str);

#include <conio.h>
void clreol(void);

#include <graphics.h>
int far getmaxx(void);
int far getmaxy(void);

#include <graphics.h>
void far moveto(int x, int y);

#include <graphics.h>
void far putpixel(int x, int y, int color);

#include <graphics.h>
void far setlinestyle(int linestyle,
                    unsigned upattern,
                    int thickness);

```

شکل (۳)


```
#include <graphics.h>
void far rectangle(int left, int top,
                  int right, int bottom);
```

```
#include <graphics.h>
void far drawpoly(int numpoints,
                  int far *polypoints);
```

```
#include <graphics.h>
void far fillpoly(int numpoints,
                  int far *polypoints);
```

```
#include <graphics.h>
void far setfillstyle(int pattern,
                     int color);
```

```
#include <graphics.h>
void far arc(int x, int y, int stangle,
             int endangle, int radius);
```

```
#include <graphics.h>
struct arccoordstype {int x, y;
                     int xstart, ystart;
                     int xend, yend
};
```

```
#include <graphics.h>
void far getarccoords(
    struct arccoordstype far *arccoords);
```

```
#include <graphics.h>
void far ellipse(int x, int y,
                 int stangle,
                 int endangle,
                 int xradius,
                 int yradius);
```

شکل (۴)

```
#include <graphics.h>
void far fillellipse(int x, int y,
                    int xradius,
                    int yradius);
```

```
#include <graphics.h>
int far getmaxcolor(void);
```

```
#include <graphics.h>
void far sector(int x, int y,
               int stangle,
               int endangle,
               int xradius,
               int yradius);
```

```
#include <graphics.h>
void far pieslice(int x, int y,
                 int stangle,
                 int endangle,
                 int radius);
```

```
#include <graphics.h>
void far bar(int left, int top,
            int right, int bottom);
```

```
#include <graphics.h>
void far bar3d(int left, int top,
              int right, int bottom,
              int depth, int topflag);
```

```
#include <graphics.h>
void far setfillpattern(char far *upattern
                      int color);
```

```
#include <graphics.h>
void far floodfill(int x, int y,
                  int border);
```

شکل (۵)

```

#include <graphics.h>
void far outtext(char far *textstring);

#include <graphics.h>
void far outtextxy(int x, int y,
                  char far *textstring);

#include <graphics.h>
void far settextstyle(int font,
                    int direction,
                    int charsize);

#include <graphics.h>
void far settextjustify(int horiz,
                      int vert);

#include <graphics.h>
void far gettextsettings(
    struct textsettingstype far *texttypeinfo);

#include <graphics.h>
struct textsettingstype {
    int font;
    int direction;
    int charsize;
    int horiz;
    int vert;
};

#include <graphics.h>
int far textheight(char far *textstring);
int far textwidth(char far *textstring);

#include <stdio.h>
int sprintf (char *buffer,
            const char *format [, argument, ...]);

```

شکل (٦)

```

#include <graphics.h>
void far setusercharsize(int multx,
                        int divx,
                        int multy,
                        int divy);

#include <graphics.h>
int far graphresult(void);

#include <graphics.h>
char *far grapherrormsg(int errorcode);

#include <graphics.h>
void far setviewport(int left, int top,
                    int right, int bottom,
                    int clip);

#include <graphics.h>
struct viewporttype {
    int left;
    int top;
    int right;
    int bottom;
    int clip;
};

#include <graphics.h>
void far getviewsettings (
    struct viewporttype far *viewport);

#include <graphics.h>
void far clearviewport(void);

#include <graphics.h>
void far cleardevice(void);

```

شكل (٧)

```
#include <graphics.h>
void far graphdefaults(void);

#include <graphics.h>
void far setactivepage(int page);
void far setvisualpage(int page);

#include <dos.h>
void delay(unsigned milliseconds);

#include <graphics.h>
void far getimage(int left, int top,
                  int right, int bottom,
                  void far *bitmap);

#include <graphics.h>
void far putimage(int left, int top,
                  void far *bitmap,
                  int op);

#include <graphics.h>
unsigned far imagesize(int left, int top,
                       int right, int bottom);

#include <graphics.h>
void far setpalette(int colornum,
                    int color);

#include <stdlib.h>
int random(int num);
```

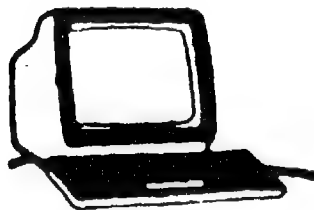
شکل (۸)

```
#include <conio.h>
int gettext(int left, int top,
            int right, int bottom,
            void *destin);
```

```
#include <conio.h>
int puttext(int left, int top,
            int right, int bottom,
            void *source);
```

```
#include <dos.h>
int int86(int int_number,
          union REGS *in_regs,
          union REGS *out_regs);
```

شکل (۹)



الملحق (د)



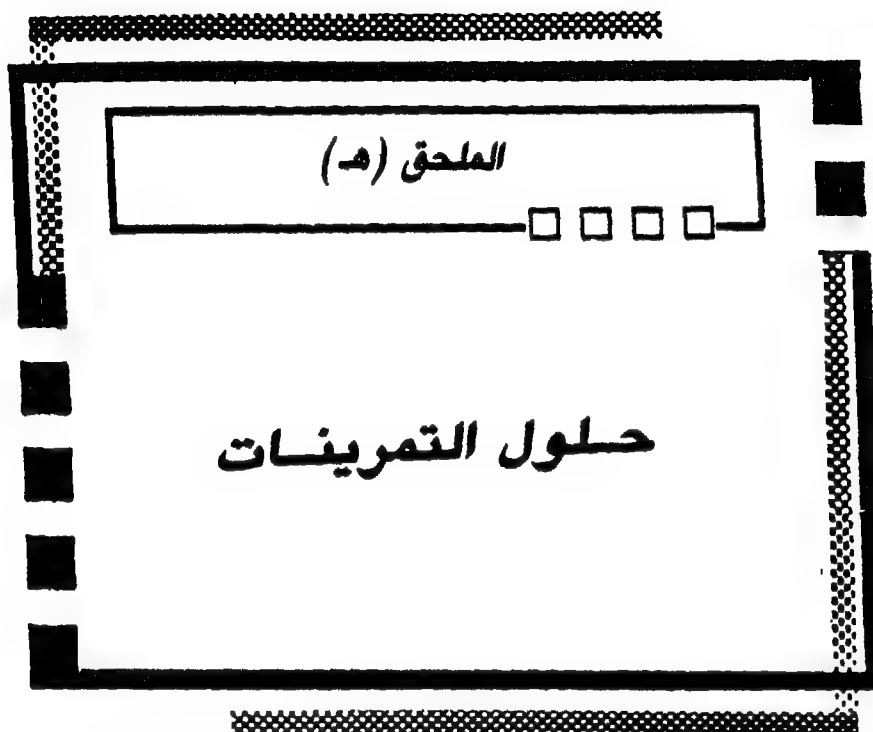
أهم خصائص لغة سي++ بمقارنتها بلغة سي (تستخدم لتحويل البرامج إلى لغة سي)

[١] استخدام الإعلانات في أى مكان فى البرنامج (للعودة إلى لغة سي ضع الإعلانات فى المقدمة) .

[٢] استخدام دالة الخرج cout الموجودة بملف العناوين "iostream.h" (للعودة للغة سي) استخدم الدالة printf وملف العناوين stdio.h .

[٣] استخدام اسم المنشأ كنمط (type) بدون كلمة struct (يلزم فى لغة سي استخدام كلمة struct مع المنشآت) .

[٤] لا يجوز تخصيص مؤشر خلال من الرصيد (void) إلى مؤشر من نوع آخر (مثل int أو char) بدون استخدام الإسقاط (casting) .



حل التدريب (١ - ١)

```
/* Program DRL1-1.cpp */
#include <graphics.h>
#include <conio.h>
main()
{
    int driver, mode;
    driver = DETECT;
    initgraph(&driver, &mode, "D:/TC/BGI");
    line(0,0,639,479);
    line(639,0,0,479);
    circle(0,0,160);
    circle(639,0,160);
    circle(0,479,160);
    circle(630,479,160);
    // Wait for a key:
    getch();
    return(0);
}
```

حل التدريب (١ - ٢)

```
/* Program DRL1-2.cpp */
#include <graphics.h>
#include <conio.h>
main()
{
    int driver, mode;
    driver = DETECT;
    initgraph(&driver, &mode, "D:/TC/BGI");
    lineto(639,0);
    lineto(639,479);
    lineto(0,479);
    lineto(0,0);
    getch();
    return(0);
}
```

حل تمرين (١-٢)

```

/* Program DRL2-1.cpp */
#include <conio.h>
#include <stdio.h>
/* Outside frame coordinates: */
#define m1 1
#define n1 1
#define m2 80
#define n2 25
/* First window coordinates: */
#define a1 20
#define b1 5
#define a2 60
#define b2 10
/* Windows definitions */
#define WINDOW1() window(a1,b1,a2,b2);
#define WINDOW2() window(a1,b1+10,a2,b2+10);
/* Prototype declaration */
void frame(int, int, int ,int);
main()
{
    char *a, *b, *c, *d;
    a="Hi There.. I am in window #1.";
    b="Press any key..";
    c="Hello again.. I am in window #2.";
    d="Hello..I am in the big frame.";
    clrscr();
    textmode(C80);
    window(m1,n1,m2,n2);
    frame(m1,n1,m2,n2);
    gotoxy(m1,n1);
    textcolor(BLUE);
    highvideo();
    cprintf("%s",d);
    gotoxy(52,24);
    cprintf("%s",d);
    gotoxy(30,22);
    textcolor(GREEN);
    highvideo();
    cprintf("%s",b);
    getch();
    WINDOW1();
    clrscr();
    textcolor(BROWN);
    highvideo();
}

```

تابع حل تمرين (٢ - ١)

```

frame(a1,b1,a2,b2);
textcolor(RED);
highvideo();
gotoxy(3,2);
cprintf("%s",a);
gotoxy(3,4);
cprintf("%s",b);
getch();
WINDOW2();
clrscr();
textcolor(MAGENTA);
highvideo();
frame(a1, b1+10, a2, b2+10);
textcolor(CYAN);
highvideo();
gotoxy(3,2);
cprintf("%s",c);
gotoxy(3,4);
cprintf("%s",b);
getch();
return(0);
}
/* frame function */
void frame(int x1, int y1, int x2, int y2)
{
    register int i;
    gotoxy(1,1);
    for (i=0; i<= x2-x1; i++)
        putchar('-');
    gotoxy(1, y2-y1);
    for (i=0; i<= x2-x1; i++)
        putchar('-');
    for (i=2; i< y2-y1; i++) {
        gotoxy(1, i);
        putchar('|');
        gotoxy(x2-x1+1, i);
        putchar('|');
    }
}

```

حل التدريب (٣-١)

```
/* Program DRL3-1.cpp */
// Vertical eccentric circles
#include <graphics.h>
#include <conio.h>
main()
{
    int driver, mode;
    int HalfWidth;
    int x, y, r;
    long loop;
    driver = DETECT;
    initgraph(&driver, &mode, "D:/TC/BGI");
    setcolor(YELLOW);
    HalfWidth = getmaxx()/2;
    // The graph:
    for (r=10; r<=200; r=r+10) {
        circle(HalfWidth, r, r);
        for (loop=1; loop<=100000; loop++);
    }
    getch();
    return(0);
}
```

حل التدريب (٣-٢)

```
/* Program DRL3-2.cpp */
// Open cone
#include <graphics.h>
#include <conio.h>
main()
{
    int driver, mode, r;
    long loop;
    driver = DETECT;
    initgraph(&driver, &mode, "D:/TC/BGI");
    setcolor(YELLOW);
    for (r=50; r<=230; r=r+10) {
        circle(r,r,r);
        for (loop=1; loop<=100000; loop++);
    }
    getch();
    return(0);
}
```

حل التدريب (٣-٣) ا

```

/* Program DRL3-3A.cpp */
// Partial donut
#include <graphics.h>
#include <conio.h>
#include <math.h>
#define PI 3.14159
main()
{
    int driver, mode;
    int HalfHeight, HalfWidth;
    double x, y, r, k, Angle, Incr;
    long loop;
    Angle = PI;           // To draw half of a donut
    Incr = 2*PI/360;
    r = 100;
    driver = DETECT;
    initgraph(&driver, &mode, "D:/TC/BGI");
    HalfHeight = getmaxy()/2;
    HalfWidth = getmaxx()/2;
    for (k=0; k<=Angle-Incr; k=k+Incr*5) {
        x=HalfWidth+r*cos(k);    // shifting X
        y=HalfHeight+r*sin(k);   // shifting Y
        circle(x,y,50);
        for (loop=1; loop<=10000; loop++);
    }
    getch();
    return(0);
}

```

حل التدريب (٣-٣) ب

```

/* Program DRL3-3B.cpp */
// Solid donut
#include <graphics.h>
#include <conio.h>
#include <math.h>
#define PI 3.14159
main()
{
    int driver, mode;
    int HalfHeight, HalfWidth;

```

```

double x, y, r, k, Angle, Incr;
long loop;
Angle = 2*PI;
Incr = 2*PI/360;
driver = DETECT;
initgraph(&driver, &mode, "D:/TC/BGI");
HalfHeight = getmaxy()/2;
HalfWidth = getmaxx()/2;
r = 100;
for (k=0; k<=Angle-Incr; k=k+Incr*5) {
    x=HalfWidth+r*cos(k);          // shifting X
    y=HalfHeight+r*sin(k);         // shifting Y
    circle(x,y,r);
    for (loop=1; loop<=10000; loop++);
}
getch();
return(0);
}

```

حل التدريب (٣-٣) ج

```

/* Program DRL3-3C.cpp */
// Full-screen downut
#include <graphics.h>
#include <conio.h>
#include <math.h>
#define PI 3.14159
main()
{
    int driver, mode;
    int HalfHeight, HalfWidth;
    double x, y, r, k, Angle, Incr;
    long loop;
    Angle = 2*PI;
    Incr = 2*PI/360;
    driver = DETECT;
    initgraph(&driver, &mode, "D:/TC/BGI");
    HalfHeight = getmaxy()/2;
    HalfWidth = getmaxx()/2;
    r = HalfHeight;
    for (k=0; k<=Angle-Incr; k=k+Incr*5) {
        x=HalfWidth+r*cos(k);          // shifting X
        y=HalfHeight+r*sin(k);         // shifting Y
        circle(x,y,r);
        for (loop=1; loop<=10000; loop++);
    }
}

```

```

    }
    getch();
    return(0);
}

```

حل التدریب (٣ - ٤)

```

/* Program DRL3-4.cpp */
// Rotating lines with diff. radii and colors
#include <graphics.h>
#include <conio.h>
#include <math.h>
#define PI 3.14159
main()
{
    int driver, mode;
    int HalfHeight, HalfWidth;
    double x, y, r, k, n, Angle, Incr;
    long loop;
    Angle = 2*PI;
    Incr = 2*PI/360;
    n = 70; // smallest length of the line
    driver = DETECT;
    initgraph(&driver, &mode, "D:/TC/BGI");
    HalfHeight = getmaxy()/2;
    HalfWidth = getmaxx()/2;
    for (r=n; r <= 3*n; r+=n) {
        setcolor(r/n+4); // Change color with changing r
        for (k = 0; k <= Angle-Incr; k = k+Incr*3) {
            x = HalfWidth + r*cos(k); // shifting X
            y = HalfHeight + r*sin(k); // shifting Y
            moveto(HalfWidth,HalfHeight);
            lineto(x,y);
            for (loop = 1; loop <= 10000; loop++);
        }
    }
    getch();
    return(0);
}

```

حل التدريب (٥-٣)

```

/* Program DRL3-5.cpp */
// Shell
#include <graphics.h>
#include <conio.h>
#include <math.h>
#define PI 3.14159
main()
{
    int driver, mode;
    int HalfHeight, HalfWidth;
    double x, y, r, k, Angle, Incr;
    long loop;
    Angle = 2*PI;
    Incr = 2*PI/360;
    r = 130;           // radius
    driver = DETECT;
    initgraph(&driver, &mode, "D:/TC/BGI");
    HalfHeight = getmaxy()/2;
    HalfWidth = getmaxx()/2;
    for (k = 0; k <= Angle-Incr; k = k+Incr*5) {
        x = 0.8 * HalfWidth + r*cos(k);           // shifting X
        y = 0.6 * HalfHeight + r*sin(k);          // shifting Y
        moveto(HalfWidth, HalfHeight);
        lineto(x, y);
        for (loop = 1; loop <= 10000; loop++);
    }
    getch();
    return(0);
}

```

حل التدريب (٦-٣)

```

/* Program DRL3-6.cpp */
#include <graphics.h>
#include <conio.h>
#include <math.h>
#include <iostream.h>
#define PI 3.14159
main()
{
    int driver, mode;
    int HalfHeight;

```



```

double x, y, Amp, w;
long loop;
driver = DETECT;
initgraph(&driver, &mode, "D:/TC/BGI");
HalfHeight = getmaxy()/2;
line(0,10,0,getmaxy()-10); // Y axis
line(0,HalfHeight,getmaxx(),HalfHeight); // X axis
for (x=0; x <= 360; x+=0.1) {
    w = 6*x*(2*PI/360); // Angular frequency
    Amp = w; // Max. Amplitude
    y = HalfHeight - Amp*sin(w);
    putpixel(x,y,YELLOW);
}
gotoxy(5,5); cout << "Growing sine wave";
getch();
closegraph();
return(0);
}

```

حل التدريب (٧-٣)

```

/* Program DRL3-7.cpp */
// Cylinder
#include <graphics.h>
#include <conio.h>
#include <math.h>
#define PI 3.14159
main()
{
    int driver, mode;
    double x1, y1, y2, r;
    long loop;
    driver = DETECT;
    x1 = 200;
    y1 = 100;
    y2 = 350;
    r = 100;
    initgraph(&driver, &mode, "D:/TC/BGI");
    setcolor(YELLOW);

    circle(x1,y1,r);
    circle(x1,y2,r);
}

```

```

for(int fi=0; fi<=180*2; fi+=10) {
    line(x1+r*cos(fi*PI/180), y1+r*sin(fi*PI/180),
        x1-r*cos(fi*PI/180), y2-r*sin(fi*PI/180));
    for(long i=1; i<=100000; i++);
}
getch();
return(0);
}

```

حل التدريب (٣-٨)

```

/* Program DRL3-8.cpp */
// Cylinder
#include <graphics.h>
#include <conio.h>
#include <math.h>
#define PI 3.14159
main()
{
    int driver, mode;
    double x1, y1, y2, r;
    long loop;
    driver = DETECT;
    x1 = 200;
    y1 = 100;
    y2 = 350;
    r = 100;
    initgraph(&driver, &mode, "D:/TC/BGI");
    setcolor(YELLOW);
    setlinestyle(DOTTED_LINE, 0, THICK_WIDTH);
    circle(x1, y1, r);
    circle(x1, y2, r);
    for(int fi=0; fi<=180; fi+=4) {
        line(x1+r*cos(fi*PI/180), y1-r*sin(fi*PI/180),
            x1+r*cos(fi*PI/180), y2-r*sin(fi*PI/180));
        for(long i=1; i<=100000; i++);
    }
    getch();
    return(0);
}

```

حل التدريب (١-٤)

```

/* Program DRL4-1.cpp */
// Filled Polygons
#include <graphics.h>
#include <conio.h>
#include <iostream.h>
//
#define NoOfPoints 5
#define NoOfCoord NoOfPoints*2
//
main()
{
    int driver, mode;
    int PointArray[NoOfCoord+2] = {100,100, 200,130
                                   ,250,190, 300,300
                                   ,140,250, 100,100};

    driver = DETECT;
    initgraph(&driver, &mode, "D:/TC/BGI");
    setcolor(YELLOW);
    // Set line style:
    setlinestyle(SOLID_LINE,0,THICK_WIDTH);
    // Draw the polygon and fill with blue HATCH pattern:
    cout << endl << "Press any key...";
    for(int i=0; i<12; i++) {
        setfillstyle(i, WHITE);
        fillpoly(NoOfPoints+1,PointArray);
        getch();
    }
    closegraph();
    return(0);
}

```

حل التدريب (٢-٤)

```

/* Program DRL4-2.cpp */
// Arcs
#include <graphics.h>
#include <conio.h>
main()
{
    int driver, mode;
    driver = DETECT;
    arccoordstype P;
    initgraph(&driver, &mode, "D:/TC/BGI");
}

```

```

int radius = getmaxy()/2;
setcolor(YELLOW);
arc(getmaxx()/2, getmaxy()/1.5,
    90, 180, radius);
getarccoords(&P);
line(P.xstart, P.ystart, P.x, P.y);
line(P.xend, P.yend, P.x, P.y);
getch();
closegraph();
return(0);
}

```

حل التمرين (٤-٣)

```

/* Program DRL4-3.cpp */
// Ellipses
#include <graphics.h>
#include <conio.h>
main()
{
    int driver, mode;
    driver = DETECT;
    initgraph(&driver, &mode, "D:/TC/BGI");
    setlinestyle(SOLID_LINE, 0, THICK_WIDTH);
    int HalfWidth = getmaxx()/2;
    int HalfHeight = getmaxy()/2;
    int xradius = getmaxx()/4;
    int yradius = xradius/2;
    setcolor(YELLOW);
    ellipse(HalfWidth, HalfHeight,
        0, 180,
        xradius, yradius);
    getch();
    closegraph();
    return(0);
}

```

حل التدريب (٤ - ٤)

```

/* Program DRL4-4.cpp */
// Pie Slice
#include <graphics.h>
#include <conio.h>
#include <iostream.h>
main()
{
    int mode, driver = DETECT;
    int HalfWidth, HalfHeight;
    int radius;
    int stangle = 0, endangle = 60;
    initgraph(&driver, &mode, "d:/tc/bgi");
    HalfWidth = getmaxx()/2;
    HalfHeight = getmaxy()/2;
    radius = HalfWidth/2;
    cout << endl << "Press any key...";
    for (int i = EMPTY_FILL; i <= CLOSE_DOT_FILL; i++) {
        setfillstyle(i, getmaxcolor());
        pieslice(HalfWidth, HalfHeight,
                stangle, endangle,
                radius);
        getch();
    }
    closegraph();
    return (0);
}

```

حل التدريب (٤ - ٥)

```

/* Program DRL4-5.cpp */
// Bar and Bar3D
#include <graphics.h>
#include <conio.h>
main()
{
    int mode, driver = DETECT;
    int HalfWidth, HalfHeight;
    initgraph(&driver, &mode, "d:/tc/bgi");
    HalfWidth = getmaxx()/2;
    HalfHeight = getmaxy()/2;
    setfillstyle(CLOSE_DOT_FILL, getmaxcolor());

```

```

bar(HalfWidth-200, HalfHeight-200,
    HalfWidth-50, HalfHeight-100);
bar(HalfWidth-200, HalfHeight,
    HalfWidth-100, HalfHeight+200);
bar3d(HalfWidth+150, HalfHeight-200,
    HalfWidth+50, HalfHeight-50, 10, 0);
bar3d(HalfWidth+50, HalfHeight+50,
    HalfWidth+150, HalfHeight+150, 30, 0);
getch();
closegraph();
return(0);
}

```

حل التدريب (١-٥)

```

/* Program DRL5-1.cpp */
// Text justification
#include <graphics.h>
#include <conio.h>
main()
{
    int driver, mode;
    driver = DETECT;
    char *string1 = "Middle of Vertical Line.";
    char *string2 = "Middle of Horizontal Line.";
    initgraph(&driver, &mode, "d:\\tc\\BGI");
    int font = TRIPLEX_FONT;
    int size = 3;
    // Justify text:
    settextjustify(1,1);
    // Set font, direction, and size:
    settextstyle(font, HORIZ_DIR, size);
    outtextxy(getmaxx()/2,40, string2);
    outtextxy(getmaxx()/2,getmaxy()-40, string2);
    // Set font, direction, and size:
    settextstyle(font, VERT_DIR, size);
    for (int i=10; i <= getmaxx(); i+=getmaxx()/4-10)
        outtextxy(i, getmaxy()/2, string1);
    getch();
    closegraph();
    return(0);
}

```

حل التدريب (٣-٥)

```

/* Program DRL5-3.cpp */
// User defined character size
#include <graphics.h>
#include <conio.h>
void square(char *);
main()
{
    int driver = DETECT, mode;
    char *string[6]= { " NORMAL",
                       " NARROW",
                       " WIDE",
                       " SHORT",
                       " HIGH",
                       " HUGE"};

    initgraph(&driver, &mode, "d:\\tc\\bgi");
    // Text style and initial zise:
    settextstyle(SANS_SERIF_FONT, HORIZ_DIR, 0);
    int x = getmaxx()*0.05;
    int y = getmaxy()*0.05;
    moveto(x,y);
    // Normal text:
    outtext(string[0]);
    square(string[0]);
    // Reduce text width by 1/2:
    setusercharsize(1,2,1,1);
    outtext(string[1]);
    square(string[1]);
    // Magnify width 4 times:
    setusercharsize(4,1,1,1);
    outtext(string[2]);
    square(string[2]);
    // Reduce height by 1/3:
    moveto(x, y+100);
    setusercharsize(1,1,1,3);
    outtext(string[3]);
    square(string[3]);
    // Increase height 4 times:
    setusercharsize(1,1,4,1);
    outtext(string[4]);
    square(string[4]);
}

```

تابع حل التدريب (٣-٥)

```
// Increase height 4 times and width 4 times:
setusercharsize(4,1,8,1);
outtext(string[5]);
square(string[5]);
// Clean up:
getch();
closegraph();
return(0);
}
// Function to draw a square around the text:
void square(char *string)
{
    int inc=10;
    int x1=textwidth(string);
    int y1=textheight(string);
    rectangle(getx()-x1, gety(), getx(), gety()+y1+inc);
    moveto(getx()+inc,gety());
}
```



حل التدريب (٧-٢)

```

/* Program DRL7-2.cpp */
// Moving arrow with an attached text
#include <graphics.h>
#include <stdlib.h>
#include <conio.h>
#include <iostream.h>
#include <dos.h>
#define UNIT 50
#define DELAY 100
void DrawArrow(int x, int y);
main()
{
    int driver = DETECT;
    int mode, errorcode;
    initgraph(&driver, &mode, "d:\\tc\\bgi");
    errorcode = graphresult();
    if (errorcode != grOk) {
        cout << "Graphics error:"
            << grapherrormsg(errorcode) << endl
            << "Make sure the path to graphics "
            << "drivers is correct." << endl
            << "Press any key to exit";
        getch();
        clrscr();
        exit(1);
    }
    void *ImagePointer;
    unsigned int size;
    int x = 0, y = getmaxy()/2;
    // Draw the arrow:
    DrawArrow(x,y);
    gotoxy(10,14);
    cout << "Press any key...";
    getch();
    // Calculate the picture size:
    size = imagesize(x,
                    y - UNIT,
                    x + 4*UNIT,
                    y + UNIT);
    // Allocate the necessary memory:
    ImagePointer = malloc(size);
    // Save the picture in memory:
    getimage(x,
            y - UNIT,

```

تابع حل التدريب (٧-٢)

```

        x + 4*UNIT,
        y + UNIT,
        ImagePointer);
// Start motion:
while (!kbhit()) {
// Erase the existing image:
    putimage(x,
        y - UNIT,
        ImagePointer,
        XOR_PUT);
// Advance the x-position one unit:
    x += UNIT;
    if (x >= getmaxx())
        x = 0;
// Put an image copy in the new position:
    putimage(x,
        y - UNIT,
        ImagePointer,
        XOR_PUT);
    delay(DELAY);
}
free(ImagePointer);
closegraph();
return(0);
}
// A function to draw the arrow:
void DrawArrow(int x, int y)
{
    setfillstyle(SOLID_FILL, RED);
    moveto(x,y);
// Draw the arrow:
    linerel(4*UNIT, 0);
    linerel(-2*UNIT, -1*UNIT);
    linerel(0, 2*UNIT);
    linerel(2*UNIT, -1*UNIT);
// Fill lower triangle with red color:
    int f1 = x + 2.5*UNIT;
    int f2 = y + 0.5*UNIT;
    floodfill(f1, f2, WHITE);
// Fill upper triangle with red color:
    f2 = f2 - UNIT;
    floodfill(f1, f2, WHITE);
}

```

حل التدریب (٧-٣)

```

/* Program DRL7-3 */
// Moving text
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <iostream.h>
#include <dos.h>
#define DELAYTIME 80
#define VTAB 1.5
#define FONTSIZE 4
//
main()
{
    int driver = DETECT;
    int mode, errorcode;
    initgraph(&driver, &mode, "d:\\tc\\bgi");
    errorcode = graphresult();
    if (errorcode != grOk) {
        cout << "Graphics error:"
            << grapherrormsg(errorcode) << endl
            << "Make sure the path to graphics "
            << "drivers is correct." << endl
            << "Press any key to exit";
        getch();
        clrscr();
        exit(1);
    }
    int x=0, y=0;
    char *string = "Moving Text";
    unsigned int size;
    void *ImagePointer;
    int Maxx = getmaxx();
    int Maxy = getmaxy();
    settxtstyle(TRIPLEX_FONT, HORIZ_DIR, FONTSIZE);
    moveto(x,y);
    // Display the string:
    outtext(string);
    // Calculate image size:
    int X = textwidth(string);
    int Y = VTAB * textheight(string);
    size = imagesize(x, y, x+X, y+Y);
    ImagePointer = malloc(size);

```

تابع حل التدريب (٧-٣)

```
// Save image:
getimage(x, y, x+X, y+Y, ImagePointer);
// Start motion loop:
while (!kbhit()) {
    putimage(x, y, ImagePointer, XOR_PUT);
    y += Y;
    if (y > Maxy-Y) {
        x += X;
        y = 0;
    }
    if (x > Maxx-X) x=0;
    putimage(x, y, ImagePointer, XOR_PUT);
    if (y < Maxy-Y)
        delay(DELAYTIME);
    else
        delay(0.1*DELAYTIME);
}
free(ImagePointer);
closegraph();
return(0);
}
```



حل التدريب (٧ - ٤)

```

/* Program DR17-4.cpp */
// Animated Blinking Star
#include <graphics.h>
#include <stdlib.h>
#include <conio.h>
#include <iostream.h>
#include <dos.h>
#define UNIT 40
#define DELAYTIME 150
void DrawStar(int x, int y);
main()
{
    int driver = DETECT;
    int mode, errorcode;
    initgraph(&driver, &mode, "d:\\tc\\bgi");
    errorcode = graphresult();
    if (errorcode != grOk) {
        cout << "Graphics error:"
            << grapherrormsg(errorcode) << endl
            << "Make sure the path to graphics "
            << "drivers is correct." << endl
            << "Press any key to exit";
        getch();
        clrscr();
        exit(1);
    }
    void *ImagePointer;
    unsigned int size;
    int x = UNIT, y = getmaxy()/2;
    int color=getmaxcolor();
    // Draw the star:
    DrawStar(x,y);
    settextstyle(TRIPLEX_FONT, HORIZ_DIR, 2);
    settextjustify(CENTER_TEXT, CENTER_TEXT);
    outtextxy(getmaxx()/2, getmaxy()/8, "Press any key...");
    getch();
    // Calculate the picture size:
    size = imagesize(x - UNIT,
                    y - UNIT,
                    x + 2*UNIT,
                    y + 2*UNIT);
    // Allocate the necessary memory:
    ImagePointer = malloc(size);

```

تابع حل التدريب (٧ - ٤)

```
// Save the picture in memory:
getimage(x - UNIT,
        y - UNIT,
        x + 2*UNIT,
        y + 2*UNIT,
        ImagePointer);
// Start motion:
while (!kbhit()) {
// Erase the existing image:
putimage(x - UNIT,
        y - UNIT,
        ImagePointer,
        XOR_PUT);
// Change color:
color=random(15)+1;
setpalette(color,random(15)+1);
// Change position:
x += UNIT;
if (x > getmaxx())
    x = UNIT;
// Paste it:
putimage(x - UNIT,
        y - UNIT,
        ImagePointer,
        XOR_PUT);
delay(DELAYTIME);
}
free(ImagePointer);
closegraph();
return(0);
}
// A function to draw the arrow:
void DrawStar(int x, int y)
{
    int color = getmaxcolor();
    setfillstyle(SOLID_FILL, WHITE);
    moveto(x,y-UNIT);

// Draw the arrow:
linerel(UNIT, 2*UNIT);
linerel(-2*UNIT, -1.5*UNIT);
linerel(2*UNIT, 0);
```

تابع حل التدريب (٧ - ٤)

```

linerel(-2*UNIT, 1.5*UNIT);
linerel(UNIT, -2*UNIT);
// Fill
setfillstyle(SOLID_FILL, color);
floodfill(x, y, color);
setfillstyle(SOLID_FILL, color-1);
floodfill(x, y-0.7*UNIT, color);
setfillstyle(SOLID_FILL, color-2);
floodfill(x+0.7*UNIT, y-0.4*UNIT, color);
setfillstyle(SOLID_FILL, color-3);
floodfill(x-0.7*UNIT, y-0.4*UNIT, color);
setfillstyle(SOLID_FILL, color-4);
floodfill(x+0.4*UNIT, y+0.1*UNIT, color);
setfillstyle(SOLID_FILL, color-5);
floodfill(x-0.4*UNIT, y+0.1*UNIT, color);
}

```



المراجع

BOOK	AUTHOR	PUBLISHER
THE C PROGRAMMING LANGUAGE	BRIAN KERNIGHAN-DENNIS RITCHIE	PRINTICE-HALL INC.
HIGH RES. COMP. GRAPHICS USING C	IAN O. ANGELL	HALSTED PRESS/JOHN WILEY
PROGRAMMING IN C	STEPHEN G. COCHAN	AUTHOR
TURBO C THE COMPLETE REFERENCE	HERBERT SCHILDT	BORLAND-OSBORNE/McGRAW-HILL
GRAPHICS PROGRAMMING IN C	ROGER T. STEVENS	M&T BOOKS
POWER GRAPHICS USING C++	KEITH WEISKAMP AND LOREN HEINY	WILEY
TURBO C++, SELF TEACHING GUIDE	BRYAN FLAMIG	JOHN WILEY & SONS, INC.
TURBO C++, USER'S GUIDE		BORLAND
TURBO C++, PROGRAMMERS GUIDE		BORLAND
THE WORLD OF C++		BORLAND
C++ TOOLKIT	JONATHAN S. SHAPIRO	PRINTICE HALL

كتب للمؤلف صدرت فى الولايات المتحدة

© Learn C in 3 days

© Learn Pascal in 3 days.

الناشر : Wordware publishing Inc.

تحت الطبع :

© C++ for windows

كتب للمؤلف فى مجال الكمبيوتر

صدرت عن مكتبة ابن سينا

١ - تحدث مع الكمبيوتر بلغة كوبول .

□ المستوى الأول .

٢ - كل شيء عن الكمبيوتر (وكتابة البرامج بلغة بيسك) .

□ مبسط للنشء ولأولياء الأمور .

٣ - تحدث الكمبيوتر بلغة بيسك .

□ حتى المستوى المتقدم من لغة بيسك يضم

اللغة القياسية قديمها وحديثها وأيضاً أشهر

طرازات لغة بيسك .

٤ - كيف يفكر الكمبيوتر .

□ خرائط التسلسل المنطقي للبرامج والنظم
الآلية وتحويل النظم اليدوية إلى آلية .

٥ - برمجة الألعاب الكمبيوترية .

□ طرق برمجة القذائف والتصادم والمؤثرات
الصوتية مشروحة بلغة الطرازات الشهيرة
للكمبيوتر المنزلي في مصر والعالم العربي
علاوة على لغة بيسك القياسية (ميكروسوفت) .

٦ - مدخلك إلى عالم الكمبيوتر - المقدمة الأساسية لعلوم الكمبيوتر .

٧ - تعلم لغة الكمبيوتر سى من خلال لغة بيسك .

□ مدخل مناسب للهواة والمحترفين لإجادة
لغة سى .

٨ - الرسم بالكمبيوتر .

□ يتناول كل ما يخص استخدام الكمبيوتر في
الرسم .. يشرح عبارات لغة بيسك القياسية
للرسم الدقيق علاوة على أهم اللهجات المنتشرة
لأجهزة الكمبيوتر المنزلي .

٩ - تحدث إلى الكمبيوتر بلغة لوجو .

□ لغة أصدقاء الروبوت ..
مع تطبيقات مختلفة في البرمجة والألعاب
باستخدام السلحفاة الشهيرة للرسم على الشاشة
(turtle graphics) المدخل المناسب للأطفال إلى
عالم الكمبيوتر .

١٠ - تحدث إلى الكمبيوتر بلغة فورتران ٧٧ .

□ مرجعك العربي في لغة فورتران يبدأ من
البدايات الأولى للغة ويصل حتى مستويات

متقدمة فى إنشاء البرامج . يضم الكتاب كل عبارات اللغة قديمها وحديثها مع تطبيقات على مختلف أجهزة الكمبيوتر .

١١- برامج والالعاب كومبيوترية مشروحة (بلغة بيسك) .

.. برامج تعليمية فوازير ألعاب حروب وقذائف ومغامرات .. علاوة على برامج الملفات ومعالجة الكلمات بلغة بيسك . على أشهر طرازات الكمبيوتر المنزلى : تكساس ، كومودور ، أتارى ، BBC ، إلكترو ، سنكلير ...

١٢- قبل ان تشتري كومبيوتر .

□ دليلك فى شراء جهاز كومبيوتر لمنزلك أو مكتبك أو محلّك التجارى ، دليلك فى التدريب إذا أردت العمل فى أحد مجالات الكمبيوتر نقد وتحليل خصائص أجهزة الكمبيوتر الشخصية والمنزلية .

١٣- تحدث إلى الكمبيوتر بلغة باسكال .

□ مرجعك العربى فى لغة باسكال ، قديمها وحديثها مع تطبيقات فى مختلف المجالات .

١٤- علم نفسك بنفسك لغات الجيل الرابع للكمبيوتر :

«ورد ستار» .

□ استخدام معالج الكلمات «ورد ستار» مع تدريبات مختلفة فى مجال السكرتارية وإدارة الأرشيف الإليكترونى .

١٥- علم نفسك بنفسك لغات الجيل الرابع للكمبيوتر :

دى بيز ٣ ، ٣+ .

□ استخدام قواعد البيانات فى تخزين

واسترجاع البيانات مع تدريبات مختلفة على
« برمجة » برنامج قواعد البيانات دي بيز ٣، ٣+

١٦- أشهر البرامج والروتينات بلغة بيسك

□ يحتوى الكتاب على مكتبة كاملة من البرامج الصغيرة التي يمكن استخدامها كوحداث بناء للبرامج الكبيرة . يشمل الرسم والموسيقى والحسابات والملفات ... إلى آخره

١٧- برامج وروتينات فرعية بلغة فورتران .

□ يضم الكتاب أكثر البرامج الفرعية انتشاراً ، والتي يمكن استخدامها مباشرة فى بناء البرامج الكبيرة . يتضمن مهارات المصفوفات والملفات وحل المعادلات التفاضلية .

١٨- علم نفسك بنفسك لغات الجيل الرابع للكمبيوتر لوتس 1-2-3 (المستوى الأول)

□ مبادئ استخدام الجدول الإلكتروني لوتس فى إعداد الموازنات ومائز الأغراض التجارية والمحاسبية . تطبيقات عملية على الموازنة والتكاليف .

١٩- علم نفسك بنفسك لغات الجيل الرابع للكمبيوتر : لوتس 1-2-3 . (المستوى المتقدم)

□ يستكمل معك الرحلة إلى المستويات المتقدمة فى البرنامج لوتس حيث يعرض طرق البرمجة باستخدام « الماكرو » وبناء قواعد بيانات لوتس . يتضمن الكتاب تطبيقات عملية شائعة .

٢٠- منافع نورتون

□ باقة من الأدوات التي تساعدك على تشخيص أعطال المعدات والبرمجيات وصيانتها باستخدام البرامج .

٢١ - فى قلب الكمبيوتر آى بى إم كسر حاجز الرهبة بينك وبين الكمبيوتر ...

□ يأخذك الكتاب فى رحلة شيقة فى قلب الكمبيوتر الشخصى ، نستعرض فيها أهم معالمه ، وطرق الفك والتركيب والضبط ، علاوة على استعراض الأعطال الشهيرة وطرق إصلاحها ، علاوة على طرق الارتقاء بمعدات الكمبيوتر .

٢٢ - علم نفسك بنفسك لغات الجيل الرابع للكمبيوتر : "دى بيز ٤"

□ يقدم لك قواعد البيانات فى ثوب جديد ، حيث تتعامل مع البرنامج من خلال الشاشات والقوائم والنوافذ . تطبيقات عملية من واقع الحياة اليومية للتعامل مع كميات هائلة من البيانات .

هذا بجانب إمكانية استخدام المهارات المختلفة للبرمجة التى عرضناها فى كتاب دى بيز ٣، ٣+.

٢٣ - تحدث إلى الكمبيوتر بلغة سى

□ مرجعك العربى فى لغة سى .
يبدأ معك الكتاب من المبادئ الأولية حتى يصل بك إلى أعلى مستويات البرمجة . يتضمن الكتاب موضوع "الملفات" بصورة وافية .

٢٤ - نظام التشغيل دوس (DOS)

من الطراز ١ إلى الطراز ٥

□ يتضمن الكتاب كل ما يلزمك من أدوات لتشغيل الكمبيوتر فى بيئة نظام التشغيل . DOS

٢٥ - نوافذ ميكروسوفت Microsoft Windows (الطراز 3.1)

□ يتضمن الكتاب كل ما يلزمك من أدوات
لتشغيل الكمبيوتر في بيئة النوافذ والاستفادة من
خصائصها على الوجه الأكمل .

٢٦- سي++ / سي++ للنوافذ / OPP (الجزء الأول) .

□ يتضمن الطرق الحديثة للبرمجة الموجهة
نحو الأهداف (OOP) كما يتضمن أساسيات
وقواعد لغة سي++ .

٢٧- سي++ / سي++ للنوافذ / OPP (الجزء الثاني) .

□ يتضمن خصائص البرمجة في بيئة نوافذ
ميكروسوفت . باستخدام المترجم :
- تيريو سي++ للنوافذ .
- (أو بورلاند سي++) لبرمجة التطبيقات
النوافذية .

وفي مجال الهندسة الكهربائية :

□ كل شيء عن الإلكترونيات .

وفي مجال قصص الخيال العلمي للشباب :

□ إعدام إنسان آلى .
□ الدخول في الثقب الأسود .
□ المعلوم والمجهول .



الفهرست

- ٥ كلمة الناشر ©
- ٧ كلمة المؤلف ©
- ٩ تنويه ©

الباب الأول : جولة التعارف

- ١٢ مفتاح ©
- ١٣ (١ - ١) الرسم في لغة سي و سي++
- ١٤ (٢ - ١) إعداد المترجم لبرامج الرسم
- ١٨ (٣ - ١) البرنامج الأول
- ١٩ الدخول في نسق الرسم ©
- ٢١ circle دالة رسم دائرة ©
- ٢٣ line دالة رسم مستقيم ©
- صور مختلفة لدالة رسم المستقيمات ©
- ٢٤ lineto, linerel
- ٢٨ (Graphic modes) أطوار الرسم (٤ - ١)
- ٣١ Setcolor استخدام الألوان (٥ - ١)
- ٣٤ استخدام الألوان مع الكارت CGA (حالة خاصة) ©
- ٣٧ detectgraph استكشاف المعدات (٦ - ١)
- ٣٩ getgraphmode استكشاف أطوار الرسم ©
- getmoderange
- ٤١ setgraphmode تغيير طور الرسم ©

الباب الثاني : العمل فى نسق الكتابة

Text Screen

- ٤٨ مفتاح ٢ - ١) نسق الرسم ونسق الكتابة
- ٤٩ (text mode and graphics mode)
- ٤٩ مفهوم النافذة ٢ - ٢) الأطوار فى نسق الكتابة
- ٥٠ الخروج من نسق الرسم إلى نسق الكتابة ٢ - ٢) الجمع بين الرسم والكتابة فى نسق الرسم
- ٥٣ (Text Modes) ٢ - ٣) طباعة النصوص فى نسق الكتابة
- ٥٧ Cprintf
cputs
- ٥٨ textcolor ٢ - ٤) استخدام الألوان فى نسق الكتابة
- ٦٢ textbackground ٢ - ٥) تلوين خلفية الكتابة
- ٦٥ textattr ٢ - ٦) تحديد لون الواجهة والخلفية معاً
- ٦٨ highvideo ٢ - ٧) تغيير درجة الإضاءة
lowvideo
normvideo
- ٧٠ getche ٢ - ٨) دوال إدخال وطباعة اللينات فى بيئة الكتابة
..... puch
- ٧١ clrscr ٢ - ٩) دالة مسح الشاشة أو النافذة
- ٨١ window ٢ - ١٠) النوافذ فى نسق الكتابة
- ٨٢ ٢ - ١١) استخدام النوافذ فى التطبيقات

الباب الثالث : نقط وخطوط ودوائر

- ٩٦ مفتاح ⑤
- ٩٧ الأشكال الهندسية (١ - ٣)
- ⑤ التعرف الأتوماتيكي على أبعاد الشاشة
- ٩٧ getmaxx, getmaxy
- ١٠٠ الأشكال الدائرية (٢ - ٣)
- ١٠٠ تحريك دائرة على خط مستقيم ⑤
- ١٠٤ تحريك دائرة على محيط دائرة أخرى ⑤
- ١٠٩ moveto الخطوط (٣ - ٣)
- ١١٢ مخروط باستخدام الخط الدائر ⑤
- ١١٣ قوقعة باستخدام الخط الدائر ⑤
- ١١٥ putpixel الرسم بالبكسلات (٤ - ٣)
- ١١٥ (sine wave) موجة جيبيّة ⑤
- ١١٨ damped sine wave موجة جيبيّة مخمدة ⑤
- ١٢٠ الأشكال المجسمة (٥ - ٣)
- ١٢٠ اسطوانة مجسمة ⑤
- ١٢٤ سطح مجسم ⑤
- ١٢٦ setlinestyle مواصفات الخطوط (٦ - ٣)
- ١٣٠ (User - defined line styles) أنواع الخطوط المبتكرة ⑤
- ١٣٢ الموجز ⑤

الباب الرابع : دوال الرسم والطلاء

- ١٣٤ مفتاح ⑤

- ١٣٥ .. rectangle (١ - ٤) رسم مستطيل
- ١٣٧ .. drawpoly (٢ - ٤) الأشكال المضلعة
- ١٣٩ .. fillpoly ● ملء مساحة الشكل المضلع
- التحكم في ألوان وأشكال المساحة المصمتة
- ١٤٠ .. setfillstyle
- ١٤٤ .. arc (٣ - ٤) رسم الأقواس الدائرية
- arccoordstype ● منشأ إحداثيات القوس
- ١٤٦ getarccoods
- ١٤٩ .. ellipse (٤ - ٤) القطع الناقص
- ١٥١ .. fillellipse, getmaxcolor ● ملء القطع الناقص
- ١٥٣ .. sector ● القطاع
- ١٥٥ pieslice ● القطاع الدائري
- ١٥٦ .. bar3d (٥ - ٤) القضبان المستوية والمجسمة
- ١٥٩ .. setfillpattern (٦ - ٤) شبكات الطلاء المبتكرة
- ١٦٢ .. floodfill (٧ - ٤) طلاء الأشكال بدالة الفيضان
- ١٦٥ .. (Fractals) (٨ - ٤) الرسم باستخدام "الفراكتلات"

الباب الخامس : أفانين الكتابة في نسق الرسم

- ١٧٨ مفتاح ●
- ١٧٩ بنطات الكتابة (١ - ٥)
- ١٨٠ outtext (٢ - ٥) دوال الكتابة على الشاشة
- outtextxy
- (٣ - ٥) التحكم في البنط والحجم واتجاه الكتابة
- ١٨٤ .. settextstyle
- ١٨٥ ● تغيير البنط
- ١٨٧ ● تكبير الحروف

- ١٩١ © اتجاه الكتابة
- ١٩٤ settextjustify © ضبط الهوامش
- ١٩٧ تدريب (٥ - ١)
- (٥ - ٤) تخزين واسترجاع أوضاع ضبط الكتابة
- ١٩٧ gettextsettings
- ٢٠٠ تدريب (٥ - ٢)
- ٢٠١ texthight (٥ - ٥) قياس اتساع وارتفاع الحرفيات
- textwidth
- ٢٠٤ setusercharsize (٥ - ٦) الأحجام المبتكرة للخطوط
- ٢٠٨ تدريب (٥ - ٣)

الباب السادس : مهارات عامة في الرسم

- ٢١٢ © مفتاح
- graphresult (٦ - ١) روتينات معالجة الأخطاء
- ٢١٣ grapherrormsg
- ٢١٦ setviewport (٦ - ٢) التوافق في نسق الرسم
- ٢٢٠ viewporttype © مُنشأ معلومات النافذة
- getviewsettings
- ٢٢١ clearviewport © مسح محتويات النافذة
- cleardevice أو محتويات الشاشة
- © إعادة التوافق إلى الوضع سابق التعريف
- ٢٢٥ graphdefaults
- ٢٣١ setactivepage (٦ - ٣) استخدام صفحات الذاكرة
- setvisualpage
- ٢٣٦ © برجة الأشكال المتحركة بتغيير الصفحات

الباب السابع : المحاكاة والأشكال الحية

Animation

- ٢٤٢ مفتاح ©
- ٢٤٣ خرائط الأعمدة الحية (١ - ٧)
- ٢٥١ `getimage` القص واللصق (٢ - ٧)
- `imageSize`
- تحديد الحيز المطلوب من الذاكرة لتخزين الصورة ©
- ٢٥٣ `imageSize`
- ٢٦٢ تدريب (١ - ٧)
- ٢٦٢ العمليات المنطقية على الألوان ©
- ٢٦٦ رسم شكل متحرك على الشاشة (٣ - ٧)
- ٢٧٢ تدريب (٢ - ٧)
- ٢٧٤ `(Moving Text)` تحريك نص على الشاشة (٤ - ٧)
- ٢٧٨ تدريب (٣ - ٧)
- ٢٧٨ `setpalette` الحركة بتغيير لوحة الألوان (٥ - ٧)
- ٢٨٤ تدريب (٤ - ٧)
- ٢٨٤ تطبيقات أخرى للوحة الألوان ©
- ٢٨٨ `gettext` القص واللصق في نسق الكتابة (٦ - ٧)
- `puttext`

الباب الثامن : برمجة الفأر الإلكتروني

Mouse Programming

- ٢٩٦ تنويه ©
- ٢٩٧ استخدام الفأر الإلكتروني (١ - ٨)

٢٩٨ Mouse, interrupts, registers	(٨ - ٢) الفأر والمسجلات وخطوط المقاطعة
٣٠٣	(٨ - ٣) دالة لشحن الفأر بالأحوال الابتدائية
٣٠٦	(٨ - ٤) دالة إظهار مؤشر الفأر
٣١٠	(٨ - ٥) دالة إخفاء المؤشر
٣١٤	(٨ - ٦) قراءة معلومات الفأر
٣٢١	(٨ - ٧) تنظيم البرنامج
٣٢٨	(٨ - ١) تدريب
٣٢٩	(٨ - ٨) الاستجابة لأزرار الفأر
٣٣٢	(٨ - ٢) تدريب
٣٣٤	(٨ - ٩) حزمة أدوات أدوات الفأر

الملاحق

٣٤٥	ملحق (أ) - أهم المصطلحات المعربة
٣٦١	ملحق (ب) - جدول الكود آسكي ASCII
		ملحق (ج) - عينات الدوال التي وردت في هذا الكتاب
٣٦٧	Function prototype
٣٧٧	ملحق (د) - أهم خصائص لغة سي++ بمقارنتها بلغة سي
٣٧٨	ملحق (هـ) - حلول التمرينات
٤٠٣	① المراجع
٤٠٤	② كتب للمؤلف
٤١٠	③ الفهرست

٩٤ / ٢٣٩٧

رقم الإيداع

977 - 271 - 0993

□ □ □ □ □ □ □ □ هذا الكتاب □ □

لا يستغنى مبرمج هذه الأيام عن تدعيم برنامج التطبيق بالرسم والألوان والخطوط ذات الأحجام المختلفة . وقد تكون الصورة في أحيان كثيرة أكثر بلاغة من شرح كلامي مُطَوَّل .

ولا عجب أن يقبل مستخدمو الكمبيوتر على "نوافذ ميكروسوفت" (Microsoft Windows) ويهاجرون إليها كبديل لنظام التشغيل "دوس" (DOS) ، فهي بيئة الرسم والألوان والأشكال الحية .

أما بالنسبة للمبرمج فإن بيئة النوافذ ومثيلاتها من البرامج النابضة بالحياة ليست إلا برنامجاً بلغة سي++. . فلغة سي++ عامرة بإمكانات الرسم التي تبدأ برسم الأشكال الهندسية البسيطة ، ثم تنتقل إلى بناء النوافذ والقوائم التي نتعامل معها باستخدام الفأر كما الأزوار ، ثم تصل في النهاية إلى مكتبة من الفصائل (classes) يستخدمها المبرمجون في بناء برامج مشابهة لبرنامج النوافذ ، أو لبرنامج المترجم بورلاند سي++ أو غيرها من البرامج النابضة بالحياة .

وهذا الكتاب يساعدك أن تضع قدمك على الطريق في عالم البرمجة باستخدام الرسم ، فهو يبدأ معك من البدايات الأولى ويقدم لك الكثير من الأدوات التي تدعم بها برامجك فمنحها قوة التأثير وبلاغة التعبير .

مهندس/إسماعيل الحسيني

صدر للمؤلف هذه الكتب عن لغتي سي ، سي++ .

- ⊙ تعلم لغة الكمبيوتر سي من خلال لغة بيسك (للهواة والمبتدئين) .
- ⊙ تحدث إلى الكمبيوتر بلغة سي .
- ⊙ سي++ / سي++ للنوافذ/البرمجة الموجهة نحو الأهداف (في جزئين)
- ⊙ برمجة الرسم بلغة سي++ . الناشر مكتبة ابن سينا

